

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF WISCONSIN

Filed/Received
06/26/2003 12:36:12 AM CDT

HYPERPHRASE TECHNOLOGIES, LLC and
HYPERPHRASE, INC.,

Plaintiffs,

v.

MICROSOFT CORPORATION,

Defendant.

Civil Action No. 02-C-0647 C

Chief Judge Barbara B. Crabb
Magistrate Judge Stephen L. Crocker

DECLARATION OF CRAIG R. SMITH

I, Craig R. Smith, an associate of the law firm Fish & Richardson P.C., counsel for Defendant Microsoft Corporation ("Microsoft") in the above-captioned matter, declare the following to be true, to the best of my knowledge, information, and belief:

I make this declaration in support of MICROSOFT CORPORATION'S MOTION FOR SUMMARY JUDGMENT OF NO LIABILITY.

1. Exhibit 1 is a true and accurate copy of U.S. Patent No. 5,895,461, issued on April 20, 1999 to Carlos De La Huerga and William E. Craig, entitled, "Method and System for Automated Data Storage Retrieval with Uniform Addressing Scheme."

2. Exhibit 2 is a true and accurate copy of U.S. Patent No. 6,272,505, issued on August 7, 2001 to Carlos De La Huerga, entitled, "Document Modification Based Hyperlink Limiting Method and Apparatus."

3. Exhibit 3 is a true and accurate copy of U.S. Patent No. 6,516,321, issued on February 4, 2003 to Carlos De La Huerga, entitled, "Method for Database Address Specification."

4. Exhibit 4 is a true and accurate copy of the Preliminary Expert Report of Dr. Mark K. Joseph, on behalf of Plaintiffs, dated March 3, 2003.

5. Exhibit 5 is a true and accurate copy of the July, 1945, article by Vannevar Bush, entitled, "As We May Think."

6. Exhibit 6 is a true and accurate copy of the 1986 article by L. Nancy Garrett, et al. entitled, "Intermedia: Issues, Strategies and Tactics in the Design of a Hypertext Document System."

7. Exhibit 7 is a true and accurate copy of a printout from The ACM Digital Library, showing that Exhibit 6 was published in the Proceedings of the 1986 ACM Conference on Computer-Supported Cooperative Work.

8. Exhibit 8 is a true and accurate copy of the 1988 article by Eve Wilson, entitled "Integrated Information Retrieval for Law in a Hypertext Environment."

9. Exhibit 9 is a true and accurate copy of a printout from The ACM Digital Library, showing that Exhibit 8 was published in the 1988 Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.

10. Exhibit 10 is a true and accurate copy of U.S. Patent No. 5,392,386, issued on February 3, 1994, to Chalas, entitled, "Method and Apparatus for Adding Functionality to Computer Programs Executing Under Graphical User Interfaces."

11. Exhibit 11 is a true and accurate copy of a set of screen shots of Turbo Pascal (version 1.5, copyright 1992) with explanatory comments.

12. Exhibit 12 is a true and accurate copy of the October 14, 1998 Final Office Action for U.S. Patent App. No. 08/727,293.

13. Exhibit 13 is a true and accurate copy of the July 21, 1998, Response A for U.S. Patent App. No. 08/727,293.

14. Exhibit 14 is a true and accurate copy of the November 19, 1998, Response B for U.S. Patent App. No. 08/727,293.

15. Exhibit 15 is a true and accurate copy of the December 3, 1998, Notice of Allowance for U.S. Patent App. No. 08/727,293.

16. Exhibit 16 is a true and accurate copy of the July 3, 2002 Response to Notice of Allowance and Preliminary Amendment for U.S. Patent App. No. 09/374,568.

17. Exhibit 17 is a true and accurate copy of the September 19, 2000, Amendment for U.S. Patent App. No. 09/112,062.

18. Exhibit 18 is a true and accurate copy of the U.S. Patent No. 5,377,323, issued December 27, 1994 to Vasudevan, entitled, "Apparatus and Method for a Federated Naming System Which Can Resolve a Composite name composed of Names from any Number of Disparate Naming Systems."

19. Exhibit 19 is a true and accurate copy of the November 18, 1998, Interview Summary for U.S. Patent App. No. 08/727,293.

20. Exhibit 20 is a true and accurate copy of the March 24, 2000 Office Action for U.S. Patent App. No. 09/112,062.

21. Exhibit 21 is a true and accurate copy of the May 9, 2000 Amendment for U.S. Patent App. No. 09/112,062.

22. Exhibit 22 is a true and accurate copy of the April, 1996 article by Bradley Rhodes, et al, entitled, "Remembrance Agent: A continuously running automated information retrieval system."


23. Exhibit 23 is a true and accurate copy of a printout from www.remem.org, page 2, showing that Exhibit 22 was published in the Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi Agent Technology in April, 1996.

24. Exhibit 24 is a true and accurate copy of the January, 1995 article by Airi Salminen, et al. entitled, "From Text to Hypertext by Indexing."

25. Exhibit 25 is a true and accurate copy of U.S. Patent No. 5,815,830, issued on September 29, 1998, to Anthony, entitled, "Automatic Generation of Hypertext Links to Multimedia Topic Objects."

I declare under penalty of perjury that the foregoing is true and correct.

Dated: June 26, 2003



Craig R. Smith

Exhibit 22

Remembrance Agent

A continuously running automated information retrieval system

(click for [RA distribution page](#))

Bradley J. Rhodes
MIT Media Lab, E15-305
20 Ames St.
Cambridge, MA 02139
rhodes@media.mit.edu

Thad Starner
MIT Media Lab, E15-394
20 Ames St.
Cambridge, MA 02139
thad@media.mit.edu

Published in *The Proceedings of The First International Conference on The Practical Application Of Intelligent Agents and Multi Agent Technology* (PAAM '96), pp. 487-495.

Abstract

The Remembrance Agent (RA) is a program which augments human memory by displaying a list of documents which might be relevant to the user's current context. Unlike most information retrieval systems, the RA runs continuously without user intervention. Its unobtrusive interface allows a user to pursue or ignore the RA's suggestions as desired.

Introduction

The rise of the Internet has prompted a flurry of systems designed to find, filter, and organize the huge amounts of information now available. Information retrieval systems have been developed for applications ranging from classification and prioritization of email (Maes 1994, Cohen 1996), filter netnews (Lang 1995), answering queries based on Usenet FAQ's (Hamond 1994), and searching the Web (Mauldin and Leavitt, 1994). A few applications have also been developed to organize more user specific information such as notes files, diaries, and calendars (Jones 1986, Lamming & Flynn 1994). However, almost all of these systems have concentrated on query-based information-retrieval-on-demand. For example, they can answer questions such as "when is that conference paper deadline?" or "who's an expert on this particular algorithm?". However, they do not help when the user does not remember enough to even know to ask a question, or what question to ask. This "associative" form of recall is what reminds a user that an important conference exists at all, or that there are references that should be mentioned in a paper which the user might have missed.

With more powerful desktop computers, most of a computer's CPU time is spent waiting for the user to hit the next keystroke, read the next page, or load the next packet off the network. There is no reason it can't be using those otherwise wasted CPU cycles constructively by performing continuous searches for information that might be of use in its user's current situation. For example, while an engineer reads email about a project an agent might remind her of project schedules, status reports, and other resources related to the project in question. When she stops reading email and starts editing a file, it should

automatically changes its recommendations accordingly.

The Remembrance Agent described in this paper performs this continuous, associative form of recall by continuously displaying relevant information which might be relevant to an individual user in their current context. It is designed with the philosophy that, as a continuously running and updating program, it should never distract from the user's primary task, but should instead only augment it. It therefore suggests information sources which may be relevant to the user's current situation in the form of one-line summaries at the bottom of the screen. Here they can be easily monitored, but won't distract from the primary work at hand. The full text of a suggestion can be brought up with a single keystroke.

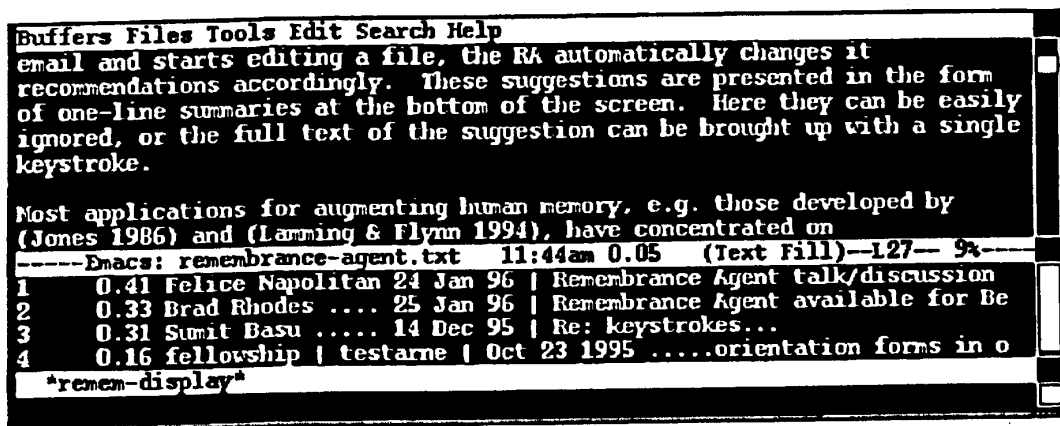


Fig. 1: A (small) screen-shot of the remembrance-agent in action

The Remembrance Agent: current implementation

The Remembrance Agent (RA) is broken into two parts. A front end continuously watches what the user types and reads, and sends this information to the back end. The back end finds old email, notes files, and on-line documents which are somehow relevant to the user's context. This information is then displayed by the front end in a way which doesn't distract from the user's primary task.

Currently, the front-end runs in elisp under Emacs-19, a UNIX based text editor which can also be used for applications such as email, netnews, and web access. It displays one-line suggestions at the bottom of the emacs display buffer, along with a numeric rating indicating how relevant it thinks the document is. These suggestions contain just enough info to relate the contents of the full document being suggested. For example, the suggestion line for a piece of email would contain who the email was from, when it was sent, and its subject-line. A notes file would contain the file-name, date last modified, the first few words of the file, and perhaps the owner of the file. With a simple key combination, the user can bring up the full text of a suggested document. The frequency with which the front end provides new suggestions, the number of suggestions, and whether to look at text notes files, old email, or other document sources is customizable for each user. The user can also customize how much of their current document the RA looks at when creating a suggestion. For example, the top display line may look at the last 500 words when making a suggestion, the two lines at the last 50 words, and the bottom line at the last 10 words. In this way, the RA can be relevant both to the user's overarching context and to any tangent they happen to be on.

The current implementation uses the SMART information retrieval program as a back end, which decides document similarity based on the frequency of words common to the query and reference documents (Salton & Lesk 1971). SMART indexes all the document sources nightly, and also supplies

relevant documents based on the "query" text supplied by the front end. While SMART is not as sophisticated as many more modern information-retrieval systems, it has the advantage that it requires no human pre-processing of the documents being indexed.

Evaluation

One difficulty in evaluating a system such as the RA is that there is a large difference between relevant suggestions (those that have a strong relation to the user's current context), and useful suggestions. For example, if the user receives email about ballroom dancing and the RA suggests an announcement for a ballroom dance event, that suggestion is highly relevant. However, if the event being announced occurred six months ago, that suggestion may be completely useless. Furthermore, a suggestion which might be useful in some situations might be more of an annoyance if the user is trying to concentrate on their primary task. For this reason, traditional methods of evaluation compare computer-generated relevancy scores with a human's opinion of relevancy (Harman, 1995) don't work well. Since the usefulness of the RA depends on the user's current task, what they already remember, how focused they are on their task, and a large number of other factors it can only be evaluated properly in the context of actual use.

The RA has been in alpha testing for about three months on a small number of platforms, so usability testing up to this point has been limited to a few users / developers. The beta-test version is now being released to a much larger user-base, so usability test will be starting shortly based both upon user surveys and logs of how often users bring up the full text of a suggestion. However, despite the small numbers of users in the alpha-testing, many lessons have been learned regarding both how the RA should be designed and to what areas it should be applied. These lessons are discussed in the next section.

Design Issues

There are three different ways of handling the timing of an automated task. The first is to perform a task only when specifically requested. Spell-checking a file and performing a web-search fall into this category. A second way is for an automated task to always lurk in the background, but to only act when a specific "trigger" occurs. Such programs include calendar programs that automatically tell you when you're going to be late for a meeting, and programs that alert you when you have new email waiting. Finally, there are tasks which are performed continuously, like a clock program or CPU-load meter. The RA falls into this third category.

There are several reasons for this design decision. As stated earlier, systems which only provide information "on request" can't help with associative recall. The user has to know that there is knowledge to be had in a particular situation. The RA therefore needs at least some ability to be proactive in its suggestions. However, unlike the calendar program that warns of upcoming meetings, it is impossible to create a back end which can reliably know when a document is useful for a user. Thus the RA will always suggest many false positives. It is also far less important that a user see an individual suggestion than it is for a calendar program. If the RA only displayed a suggestion when the relevance passed a certain threshold, the users' attention would be drawn away from their primary task whenever a new suggestion appeared. With the high ratio of false-positives, this would rapidly become a prohibitive distraction.

For these reasons the RA's suggestions are kept unobtrusive. There should be no (or at least minimal) cost to the user to see a suggestion and ignore it if deemed not useful at the time. Suggestions are

therefore kept to a single line each, and are always printed at the bottom of the text-editor window. The full display area is also limited to a maximum of 9 lines, though it defaults to operating with only three or four. Finally, no color cues or highlights were used in the suggestion-display area, and the suggestions are only changed every 10 seconds or so to further avoid distraction from the primary task. However, suggestions must also pack enough information to allow users to judge their usefulness quickly. Especially important is a date field, as the age of a document is often the best indicator of whether it is useful in a given situation. In some cases, the description line supplies desired information directly without ever needing the full document. For example, it might provide the spelling of the last name of someone who recently sent email.

While suggestions are unobtrusive, it must be trivial both to access the entirety of a suggested document and to return to the primary document once the new one has been viewed. Anecdotal evidence suggests that any more than a second or two delay in bringing up a document is a serious barrier to regular use. In the current implementation, "control-c" and the display line-number brings up the suggested document. It is similarly easy to get back to the primary document.

Using the system has shown that suggestions are much more useful when the document being suggested only contains one "nugget" of information, and when that nugget is clearly displayed on its one-line description. This "less is more" approach solves several problems. First, it allows the user to tell what a suggestion contains from its description without having to peruse the entire document. Second, a document with only one primary point is more likely to be a good hit. Documents which address several issues will rarely match the user's situation exactly, but will often partially match. Finally, if a suggested document is read, the shorter it is the quicker the user can get on with their primary work. Email and short notes files seem to be a very good length for RA suggestions. In future versions of the RA, longer documents will be broken up into pieces during indexing so that a suggestion can relate to a particular piece.

Personal email and notes files are a good source of documents for other reasons as well. Most importantly, they are both pre-personalized to the individual user and automatically change as the user's interest changes. This is one of the more important features of the RA, because it means that through proper choice of data to index the RA automatically adapts to the individual user. This guarantees that the pool of suggestions is much richer in potentially interesting information than, for example, the Web, and also helps solve many synonym problems. For example, when an AI researcher mentions "agents" in a paper, that person's RA will suggest work on autonomous agents. A chemist's RA, on the other hand, will bring up lists of various solvents and other chemical agents.

Some of the future applications discussed in the next section require using non-personalized indexes, which is analogous to using another person's memory. However, there are disadvantages beyond those discussed above. For example, it is harder to judge the content of a suggestion from its one-line summary when the user is less familiar with it. The user might also not be able to properly judge the validity of an unfamiliar suggestion. For example, they might not know that a set of instructions were made in jest, or were in a subsequent message shown to be incorrect. Similarly, documentation or lectures geared towards one kind of audience may be inappropriate for another, so it helps when the original context is well understood. Finally, there are privacy issues when using someone else's memories. Even if one accesses someone's email files with express permission, the people who sent that person mail might not have approved.

Future Applications

Wearable computing

When running on a desktop computer, an RA can only guess the user's context based on the document they are reading or editing. However, the advent of wearable computing (Starner et al. 1995) will allow RAs to work with much more information and many more situations. Global Positioning Systems (GPS) will let the RA know where the user is, while camera and face recognition will let it know who they're talking to. With this extra information, a (greatly enhanced) RA could know that it is around lunch time, that according to camera input the user is with her lunch date, according to her appointment book she has an appointment with her boss in an hour, and according to her GPS she's downtown. From this information it could recommend several good restaurants known for their fast service that are within a few blocks of downtown, which would be agreeable to her lunch date as well.

Reference advisor for technical papers

Another application currently being pursued is to have the RA suggest technical reports on a given subject. When it recommends other researchers' papers, these could be referenced or tagged for later reading. When it recommends one's own old papers, these can be scanned for similar material which might be used in the new paper. Such a system could also recommend conferences where the call-for-papers is similar in content to one's own paper.

Knowledge transfer

One of the large difficulties facing industry is bringing new members of an existing work-group up to speed quickly. If a work-group created its own knowledge-base, new members could access the group Remembrance Agent. This would not only give the employee access to the group knowledge itself (as would any database), but also to the meta-knowledge of when particular information is relevant or valuable. Similar applications would exist wherever just-in-time training is required. While this application uses a knowledge base not familiar to the user at first, they will not suffer too greatly from the lack-of-context problem discussed above because the knowledge is focused on their current situation, namely their new position within the group. Any context they do not yet know, they need to learn anyway, and the RA will help them learn it.

Automatic Hypertexting

In the past year several on-line magazines have appeared, such as HotWired (HotWired), many of which have paper counterparts. One of the values added by the on-line versions of these magazines are hyper-linked text, where a reader can click on a word and get more information on that subject. A future RA could conceivably automatically turn normal email, netnews, or papers into hyper-linked documents, automatically linking hot-words to relevant background information.

Background checker

Another Web-based RA could perform background checks on people sending mail, referenced in papers, or recognized by a wearable-computer-mounted camera. Such an RA could automatically provide information on a person's employer, job title, phone numbers, and profiles of their interests based on newsgroups they frequent. At the click of a button, the user could access that person's home-page for even more information.

Other future work

The current RA is only personalized through using personal data as the suggestion pool, and by reindexing that data every night. However, it cannot learn which of these suggested documents are

actually useful. To solve this problem, the back-end is being provided with an algorithm which learns based on which suggestions the user asks to have displayed. This way useless suggestions can be culled out over time and new ones tried.

Related work

While most information retrieval efforts have focused on query-response tasks (Harman, 1995), a few systems exist which continuously present the user with suggested web-documents based on the user's interests and the web-document currently being viewed. WebWatcher (Armstrong, Freitag, Joachims, and Mitchell, 1995) is one such system, though it requires explicit interaction to indicate interest in certain topics. Another continuously proactive web-browsing system is Letizia (Lieberman, 1995), which bases its suggestions entirely on the user's past browsing activity and links from their current page. Letizia suggests pages within a few links of the current page, and brings them up in a separate window for the user to view or ignore as required.

References

- R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell, 1995. WebWatcher: A Learning Apprentice for the World Wide Web, in AAAI Spring Symposium on Information Gathering, Stanford, CA, March 1995.
- W. Cohen, 1996. Learning Rules that Classify E-mail, in AAAI Spring Symposium on Machine Learning in Information Access, Stanford, CA, March 1996.
- D. Harman (Ed.). (1995). The Third Text REtrieval Conference (TREC-3). National Institute of Standards and Technology Special Publication 500-225, Gaithersburg, Md. 20899.
- K. Hammond, R. Burke, and K. Schmitt, 1994. A Case-Based Approach to Knowledge Navigation, in Working Papers of the AAAI Workshop on multi-media and artificial intelligence, July 1994.
- HotWired. <http://www.hotwired.com/>
- W. Jones, 1986. On the Applied Use of Human Memory Models: the Memory Extender Personal Filing System. The International Journal of Man-Machine Studies 25, 191-228
- M. Lamming and M. Flynn, 1994. " Forget-me-not:" Intimate Computing in Support of Human Memory. In Proceedings of FRIEND21, '94 International Symposium on Next Generation Human Interface, Meguro Gajoen, Japan.
- K. Lang, 1995. NewsWeeder: Learning to filter netnews, in Machine Learning: Proceedings of the Twelfth International Conference, Lake Tahoe, CA, 1995.
- H. Lieberman, Letizia: An Agent That Assists Web Browsing, International Joint Conference on Artificial Intelligence, Montreal, August 1995.
- P. Maes, 1994. Agents that Reduce Work and Information Overload. Communications of the ACM 37 (7):30-40.
- M. Mauldin and J. Leavitt, 1994. Web Agent Related Research at the Center for Machine Translation.

Remembrance Agent: A continuously running automated information retrieval system Page 7 of 7

Presented at the SIGNIDR meeting, August 4, 1994, McLean, Virginia. <http://www.lycos.com/>

G. Salton, ed. 1971. The SMART Retrieval System -- Experiments in Automatic Document Processing. Englewood Cliffs, NJ: Prentice-Hall, Inc.

T. Starner, S. Mann, B. Rhodes, J. Healey, K. Russell, J. Levine, and A. Pentland, 1995. Wearable Computing and Augmented Reality, Technical Report, Media Lab Vision and Modeling Group RT-355, MIT

C. Wickens, 1992. Engineering Psychology and Human Performance. Scott Foresman Little Brown.

Exhibit 23

The Remembrance Agent

Because serendipity is too important to be left to chance...

[\[Download\]](#) [\[Papers\]](#) [\[License\]](#)

What is the Remembrance Agent?

The Remembrance Agent (Remem) is an Emacs plug-in that watches over your shoulder and suggests information relevant to what you're reading or writing. While search engines help with direct recall, Remem is a tool for *associative* memory. Suggested documents are displayed in a buffer at the bottom of your Emacs window, and are updated every few seconds based on the last hundred or so words surrounding the cursor. Documents are pulled from your own text documents, and Remem's internal indexer can parse email archives, HTML, LaTeX and plain-text documents. It runs under most Unix systems (and maybe even properly souped-up Mac or Windows) and both Emacs and XEmacs.

Sign up for future announcements

To receive announcements of upcoming releases, please send mail to remem-announce-request@remem.org.

Download Current Release (2.11)

Below is the version 2.11 release of Remem (tarred and gzipped). It should compile on any Unix platform (let us know if it doesn't), but you'll need Emacs-20 or XEmacs-20 to run the front-end. I've heard it also compiles on various flavors of Windows using Cygwin, though there are still issues with getting the front end and back end talking to each other.

Remembrance-Agent V. 2.11 (August 6, 2001):

[[tar-gz](#) | [i386-RH7.x-rpm](#) | [i386-RH6.x-rpm](#) | [srpm](#) | [debian package](#)]

Changed from v. 2.10

- Removed pcre from the source distribution. I was including PCRE 1.09, and it's up to 3.4 now, plus the library is often included with other programs (e.g. kdesupport). You can get PCRE from www.pcre.org or Sourceforge.
- Changed parse-text so it indexes words with more than one punctuation mark after them (like "doh!!!" or "really?!?" or "hmm...")
- Updated the email_body_filter_regexp filter to only look at the first 100 lines of a header. This keeps pcre from blowing up in the rare case wehre you have an email body with thousands of lines, all indented so it looks like a single header.
- Moved the change log text out of savant.h and into ChangeLog, so people can figure out where to look.
- Removed the code in rmain.c that changed relative pathnames to absolute pathnames, as it doesn't seem to matter for Unix and Windows was having trouble.
- Updated the man pages.

Known bugs/issues

- ra-index and ra-retrieve now work under Windows, but there's still trouble with Emacs and remem.el communicating with the ra-retrieve process, at least in XP and NT. Thanks to you guys, this is getting close.
- Dates are not currently indexed, so a mouseover-query on a date doesn't produce any suggestions.
- No machine learning for different field weights

- Doesn't handle gzipped files
 - Requires GNU make, at least on some platforms, to compile.
-

Papers on the Remembrance Agent:

- Just-in-time information retrieval agents, Bradley Rhodes and Pattie Maes, *IBM Systems Journal special issue on the MIT Media Laboratory*, Vol 39, Nos. 3 and 4, 2000 pp. 685-704.
 - Just-In-Time Information Retrieval. Bradley J. Rhodes. Ph.D. Dissertation, MIT Media Lab, May 2000. Also available in [PDF](#)
 - Margin Notes: Building a Contextually Aware Associative Memory (html), Bradley J. Rhodes, to appear in *The Proceedings of the International Conference on Intelligent User Interfaces (IUI '00)*, New Orleans, LA, January 9-12, 2000.
 - The Wearable Remembrance Agent: A system for augmented memory (html), Bradley Rhodes, in *Personal Technologies Journal Special Issue on Wearable Computing*, Personal Technologies (1997) 1:218-224. See also the earlier version from *The Proceedings of The First International Symposium on Wearable Computers (ISWC '97)*, Cambridge, Mass, October 1997, pp. 123-128.
 - The Remembrance Agent: A continuously running automated information retrieval system, Bradley Rhodes and Thad Starner, *The Proceedings of The First International Conference on The Practical Application of Intelligent Agents and Multi Agent Technology (PAAM '96)*, London, UK, April 1996, pp. 487-495.
-

RA License info

As of version 2.09, the RA is released under the [GNU General Public License \(GPL\)](#). For commercial licensing under other terms, please consult the MIT Technology Licensing Office. If you have other questions, feel free to email me at rhodes@remem.org.

The RA is maintained by [Bradley Rhodes](#) (rhodes@remem.org). Let me know what you think, and as usual please send all bug reports, suggestions, and comments to ra-bugs@remem.org.

Thanks, and watch here for future releases! -- Brad

Last modified: Mon Sep 2 17:31:04 PDT 2002

Exhibit 24

From Text to Hypertext by Indexing

AIRI SALMINEN

University of Jyväskylä

and

JEAN TAGUE-SUTCLIFFE and CHARLES MCCLELLAN

The University of Western Ontario

A model is presented for converting a collection of documents to hypertext by means of indexing. The documents are assumed to be semistructured, i.e., their text is a hierarchy of parts, and some of the parts consist of natural language. The model is intended as a framework for specifying hypertextual reading capabilities for specific application areas and for developing new automated tools for the conversion of semistructured text to hypertext. In the model, two well-known paradigms—formal grammars and document indexing—are combined. The structure of the source text is defined by a schema that is a constrained context-free grammar. The hierarchic structure of the source may thus be modeled by a parse tree for the grammar. The effect of indexing is described by grammar transformations. The new grammar, called an indexing schema, is associated with a new parse tree where some text parts are index elements. The indexing schema may hide some parts of the original documents or the structure of some parts. For information retrieval, parts of the indexed text are considered to be nodes of a hypergraph. In the hypergraph-based information access, the navigation capabilities of hypertext systems are combined with the querying capabilities of information retrieval systems.

Categories and Subject Descriptors: F.4.3 [Mathematical Logic and Formal Languages]: Formal Languages—*classes defined by grammars or automata*; H.2.1 [Database Management]: Logical Design—*data models*; H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing—*indexing methods*; H.3.2 [Information Storage and Retrieval]: Information Storage; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—*retrieval models*

General Terms: Design, Management, Theory

Additional Key Words and Phrases: Constrained grammars, grammars, hypertext, properties, structured text, text entities, text types, transient hypergraphs

This research was supported by The Academy of Finland and The Natural Sciences and Engineering Research Council of Canada.

Authors' addresses: A. Salminen, University of Jyväskylä, Department of Computer Science and Information Systems, Seminaarinkatu 15, SF-40100 Jyväskylä, Finland; email: airi@cs.jyu.fi; J. Tague-Sutcliffe and C. McClellan, School of Library and Information Science, University of Western Ontario, London, Ontario N6G 1H1, Canada.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1995 ACM 1046-8188/95/0100-0069\$3.50

ACM Transactions on Information Systems, Vol. 13, No. 1, January 1995. Pages 69–99.

1. INTRODUCTION

We present a grammar/hypergraph model for converting a collection of documents to hypertext by means of indexing and for subsequent accessing of documents. There are basically two different approaches to creating hypertext. Either the text is originally designed to be handled as hypertext, or some existing text is converted to hypertext. The conversion is not always a straightforward process. Studies of such a conversion, undertaken for the *Oxford English Dictionary* [Raymond and Tompa 1988], *Engineering Data Compendium* [Glushko 1989], report abstracts, and university course catalogs [Furuta et al. 1989a], found that the process may be costly and time consuming.

1.1 Overview of the Model

The model can be used as a framework for specifying the conversion of text to hypertext and for developing new automated tools for this conversion. The model uses the same assumption as the methodology developed by Furuta et al. [1989b]: the source text is consistently structured. By *consistently structured* we mean that it is possible to define the structure by means of a formal grammar and use the grammar for developing a parser. Many documents currently created are already associated with a document description corresponding to a grammar. Standards like SGML [Goldfarb 1990], ODA [Appelt 1991], EDIFACT [ISO 1988], and ASC X12 [DISA 1990] all associate documents with document descriptions, and there are parsers available for documents written using these standards.

To describe the indexing, the model follows the approach of Burkowski [1992] and Macleod [1990; 1991], who regard index elements as parts in a hierarchic text structure. The relationship of the source text and the indexed target text is specified by grammar transformations. In the hypertextual information access, the hierarchical structure of the indexed text is associated with a separate access structure [Salminen and Watters 1992]. The access structure consists of a set of parts from the hierarchical structure and links between the parts. The links may be static or dynamic. The static links are created at the preprocessing phase; the dynamic links are created by a user query. This approach (i.e., that the user specifies a set of text elements by means of a query, and the system then dynamically creates links between the elements) has been presented earlier in Shepherd et al. [1990] and Watters and Shepherd [1990].

The model supports the dynamic nature of hypertext, but also clearly connects the dynamically created access structure to the static text structure. The static component of the hypertext is created in a preprocessing phase, based on the conversion specification made by a document analyst or a system designer. The specification consists of the following steps:

- (1) Define the hierarchical structure of the source text by a grammar.

ACM Transactions on Information Systems, Vol 13, No. 1, January 1996.

- (2) Define the indexing.
 - (i) Describe the relationship of the source text to the indexed text.
 - (ii) Define the special properties of index elements.
- (3) Define the static links.

The result of the preprocessing is a hierarchically structured, indexed text with the specified static links.

During browsing, the end user creates a current access structure. Two kinds of parts are linked during browsing: the index elements and the text elements displayed as windows. For creating the dynamic links, the user specifies

- (a) a set of index elements, and probably
- (b) an order for the elements,
- (c) the type of text parts displayed as windows, and
- (d) the layout for windows.

For (b)–(d) there may be some default specifications. Specifications (a) and (b) cause the creation of dynamic links between the specified index elements, in the specified order. Specification (c) creates links automatically between the windows containing specified index elements.

1.2 Related Work

1.2.1 Converting Text to Hypertext. A methodology for turning text into hypertext is described in Furuta et al. [1989b], based on the assumption that the source consists of regularly and consistently structured fragmented information. The target hypertext system is Hyperties [Marchionini and Shneiderman 1988], but the authors state that the framework is general enough to be applied to many hypertext systems. The methodology is a framework to guide the automatic transformation of a set of documents into hypertext form. A translator-generator system to automate the conversion is described in Stotts and Furuta [1990]. The target is assumed to consist of an ordered set of distinct text fragments. In the case of Hyperties, the fragments are called articles. The preprocessing phase includes the creation of a global index identifying all articles and listing for each article the article's title, a short description, and synonyms. The synonyms are the strings used to refer to the article within other articles. Automatic conversion creates links from the references to the article, from the article to the indices, and from an article to the preceding and following articles in each of the indices. These links are static and are always available during the browsing phase. String search in the Hyperties system can be considered as a kind of dynamic linking [Faloutsos et al. 1990].

Some hypertext systems provide special means for converting prewritten documents to hypertext. SuperBook [Egan et al. 1989] permits preprocessing

ACM Transactions on Information Systems, Vol. 13, No. 1, January 1995.

of conventional documents and then browsing them. The computer-readable source text is written in a standard text markup language such as Troff, Scribe, or Interleaf. Source documents are structured as a sequence of text fragments, each beginning with a heading. The headings are used to create the table of contents, as well as links from the table of contents to chapters and to sections. Hyperties also allows importation of articles.

GUIDEX is a system for converting existing documents to the GUIDE hypertext system [Ritchie 1989]. The source documents may be written by different text processing systems. The analysis of a document is based on the document's appearance. The structural elements of the document are identified, and the document is converted to an SGML-compatible tagged form. The GUIDE document is then created from the intermediate form using rules written by a document analyst. The conversion of legal documents to GUIDE hypertext is described in Wilson [1990].

DynaText is a system to convert different text structures to hypertext [DeRose 1990]. *DynaText* converts SGML documents to electronic books that may contain links within and among the books. The form in which SGML documents are displayed on the screen is defined on stylesheets; definition of different views through which text is displayed is also possible. Hypertext links to figures, tables, and footnotes associated with a document are shown by SGML markup in the document. Links to other documents are indicated by special document elements and corresponding stylesheet entries for the elements.

1.2.2 Indexing. Indexing is an important method for increasing the information retrieval capabilities of documents, especially documents containing natural language. Indexing is usually described from the implementation point of view, for example, by giving an algorithm for the indexing [Fawcett 1989; Salton 1981; Salton and McGill 1989].

Indexing consists of assigning identifying content indicators, called *index terms* (or phrases), to text items. These terms are then used to access the items, usually by means of an index table in the form of an inverted file. In the simplest indexing approach, words in a document are chosen as index terms and assigned to the document. The words in the document are called *index elements*. In information retrieval, the reader retrieves by an index term *i* all documents containing the index element *i*, or more generally, by a word *w* all index elements *matching w*. In the case of legal documents especially, single-term automatic indexing has sometimes proved unsatisfactory [Blair and Maron 1985]. The single-term indexing method has been refined by using term phrases instead of single terms [Gonnet 1983; 1987; Salton 1988].

In SuperBook, Hyperties, GUIDEX, and *DynaText*, preprocessing of documents involves indexing. In SuperBook, Hyperties, and *DynaText*, indexing is applied to whole text. In GUIDEX, the preprocessor creates a catalog card with fields such as document type, date, author, key words, etc., for each document. Each document may be indexed by any of, or combinations of, the catalog card fields. In all of the previous systems, the underlying hypertext

ACM Transactions on Information Systems, Vol. 13, No. 1, January 1995

model includes indexing and dynamic linking. During the browsing phase, the user specifies a set of index elements, and the system creates links automatically among the elements. Automatic generation of hypertext based on content analysis also requires some kind of text indexing. For example, in Bernstein [1990] and Salton et al. [1994], dynamic links are created between text excerpts based on similarity comparisons.

In database systems, the indexing is usually a process that is hidden from the end users. The purpose of indexing is to increase the efficiency of data management—it does not affect the result of queries. On the other hand, in text retrieval systems, the way text is indexed affects both search performance and result. A careful specification of indexing is therefore an important basis for an accurate description of the system's retrieval capabilities.

1.3 Project Goals

Our main goal is to develop a means for specifying hypertextual interfaces for specific application areas. We use the model to build a hypertext interface for the Canadian Patent Reporter¹ database. The project is briefly described in Tague et al. [1991]. The implementation is intended for law libraries. The Canadian Patent Reporter consists of full text reports of legal cases dealing with patents and trademarks, and it is an important source of information for lawyers and law students. One of the reasons why we are not using an existing hypertext system is that the capabilities for tailoring hypertext for a specific application area in current systems are very limited [Halasz 1988].

The transformation model associates a grammar as a schema both with the source and target text. In database modeling, a schema is a means for developing a model for the data management of a specific application from a general data model. The schema is used to transmit information from the database designer to the database user. The information given in the schema is then used to formulate queries or to expedite browsing [Tsichritzis and Lochovsky 1982]. Our model is proposed as a general model by which application-specific models for hypertext can be described, starting from a collection of existing structured documents. *DynaText* is a system following our model in many respects. It allows the conversion of different text structures through indexing to hypertext. Since *DynaText* hypertext consists of SGML documents, the document structures for a specific application area may be defined by SGML Document Type Definitions. However, the way text is indexed is almost the same for all documents (stop words may be defined), and the information retrieval is tied to the book metaphor.

The rest of the article is organized as follows. Section 2 describes context-free grammars and gives the definitions of the basic notions we will use: text types, values, parts, properties. Section 3 introduces our sample text, the Canadian Patent Reporter, and describes it as a source for the indexing transformation. Section 4 shows how the indexing is specified by grammar

¹ Canada Law Book Limited, published in three series during the years 1942–1993.

ACM Transactions on Information Systems, Vol. 13, No. 1, January 1995.

transformations, resulting in an indexing schema. In Section 5, a parse tree for an indexing schema will be considered as an indexed text database. A database state will be defined as a hypergraph whose nodes consist of parts of the parse tree. Section 6 discusses data access operations for indexed text databases. And Section 7 outlines the implementation of the hypertext interface for the Canadian Patent Reporter.

2. A GRAMMAR-BASED TEXT MODEL

2.1 A Context-Free Grammar as a Schema for Text

The use of a context-free grammar as a text structure definition method has been well studied, and there are many tools and techniques available for handling such text [Aho and Ullman 1972; Furuta and Stotts 1988; Gonnet and Tompa 1987; Gyssens et al. 1989; Kilpeläinen et al. 1990; Kuikka and Penttonen 1993; Macleod 1990; 1991]. SGML is an ISO standard for associating a document with a definition which corresponds to a context-free grammar [Goldfarb 1990]. Thus all SGML applications deal with text defined by a grammar. Grammars are designed for describing typical features of text: hierarchical structure, order, optionality, alternatives, recursive structures.

Formally, a context-free grammar is defined as follows:

Definition. A context-free grammar is a 4-tuple (A, N, P, s) , where

- (a) A is a set of *terminal symbols*,
- (b) N is a set of *nonterminal symbols*,
- (c) P is a set of *productions*, and
- (d) s is a distinguished nonterminal in N , called the *start symbol*.

The productions are of the form $t ::= \alpha$, where t is called the left side and α the right side of the production. The left side is a nonterminal symbol; the right side may contain nonterminal symbols, terminal symbols, and metasymbols (α may also be empty). The metasymbols are used as operators to indicate iteration, alternatives, and optionality. Iteration is denoted by $*$ (zero or more times) and $+$ (one or more times), optionality by square brackets, and $|$ separates alternatives. The parentheses are used for grouping, i.e., to show the operand of an operator and the order in which operators are applied. Terminal symbols are written between primes. A production whose left side is t is called a *t-production*.

A context-free grammar defines a formal language by specifying the allowed symbols that can be used (by the terminal symbols), and the ways these symbols can be combined to build a legal character string of the language. Such a string can be derived from the start symbol s of the grammar: the derivation starts from s and ends to the string. Let $G = (A, N, P, s)$ be a context-free grammar.

ACM Transactions on Information Systems, Vol 13, No. 1, January 1995.

Vastaaottaja: Marja
 Lähettaja: Kerttu
 Päiväys: 12/08/89
 Kokous 14.9.

Vastaaottaja: Kerttu
 Lähettaja: Marja-Kerttu
 Päiväys: 05/03/90
 Lukema oli 12/89.

Fig. 1. A collection of messages.

Vastaaottaja: Kerttu
 Lähettaja: Marja
 Päiväys: 10/12/89
 Matkustan huomenna Turkkii. Lippu maksol \$800.

Definition. A derivation is a sequence $\beta_1 \Rightarrow \dots \Rightarrow \beta_n$ where

- (a) $\beta_1 = s$,
- (b) β_n is a string of terminal symbols, and
- (c) β_i , $1 < i < n$, is a string consisting of terminal and nonterminal symbols;
- (d) each β_i , $1 < i \leq n$, is derived from β_{i-1} such that a nonterminal symbol occurrence t in β_{i-1} has been replaced by a variant of the right side of a t -production. The variant is produced as follows:
 - (i) An iteration indicated by $+$ is replaced by one or more of its operands.
 - (ii) An iteration indicated by $*$ is replaced by zero or more of its operands.
 - (iii) Alternatives are replaced by one of them.
 - (iv) Optionality is either replaced by its operand or deleted with the operand.

These replacements are conducted until all nonterminal symbols and meta-symbols are finally deleted.

Even though a context-free grammar defines a formal language, a text instance of the grammar is often semistructured, meaning that some of the parts contain natural language. In the grammar, the natural language components may be specified, for example, as a sequence consisting of words and delimiters.

Example. Suppose there is a collection of messages such as might be sent by an electronic mail system. Some examples of messages are shown in Figure 1.

76 . Airi Salminen

- (1) collection ::= (message '\n' '\n')+
- (2) message ::= 'Vastaanottaja: ' receiver '\n'
 'Lähettilää: ' sender '\n'
 'Päiväys: ' date '\n' content
- (3) sender ::= person
- (4) receiver ::= person
- (5) date ::= day '/' month '/' year
- (6) person ::= first_name
- (7) first_name ::= letter+ ['- ' letter+]
- (8) content ::= phrase
- (9) phrase ::= [delimiter] word (delimiter word)* [delimiter]
- (10) delimiter ::= delim_char+
- (11) word ::= (letter | digit)+
- (12) day ::= '01' | '02' | ... | '31'
- (13) month ::= '01' | '02' | ... | '12'
- (14) year ::= '88' | '89' | '90'
- (15) delim_char ::= ' ' | ',' | '.' | '\$' | ...
- (16) letter ::= 'A' | 'a' | 'B' | 'b' | ...
- (17) digit ::= '0' | '1' | ... | '9'

Fig. 2. A grammar for a collection of messages.

If the reader of the messages does not understand Finnish, she or he will not understand much of the content. However, by considering the text as a formal language, and by associating a description of the structure with the text, it is possible to transmit in the text some information to people who do not understand Finnish. Even if the messages are written in the language of the reader, the description of the message structure by a grammar can help the reader in data access. Figure 2 shows a context-free grammar for the collection of messages. In the grammar, the symbol ' \ n ' denotes a new line character. The start symbol of the grammar is collection.

The grammar specifies the character strings belonging to the language by which a collection of messages may be written. In Figure 1, a string of the language has been printed. In the printing, each character ' \ n ' remains invisible and causes the next character to be written at the beginning of a new line. The grammar indicates that, in Figure 1, each of the three four-line strings separated from each other by an empty line is a message. In the first message the substring "Marja" is the receiver; "Kerttu" is the sender; "12/08/89" is the date; and the fourth line is the content of the message. Both the sender and the receiver are represented by the first name of a person. For the

ACM Transactions on Information Systems, Vol. 18, No. 1, January 1995.

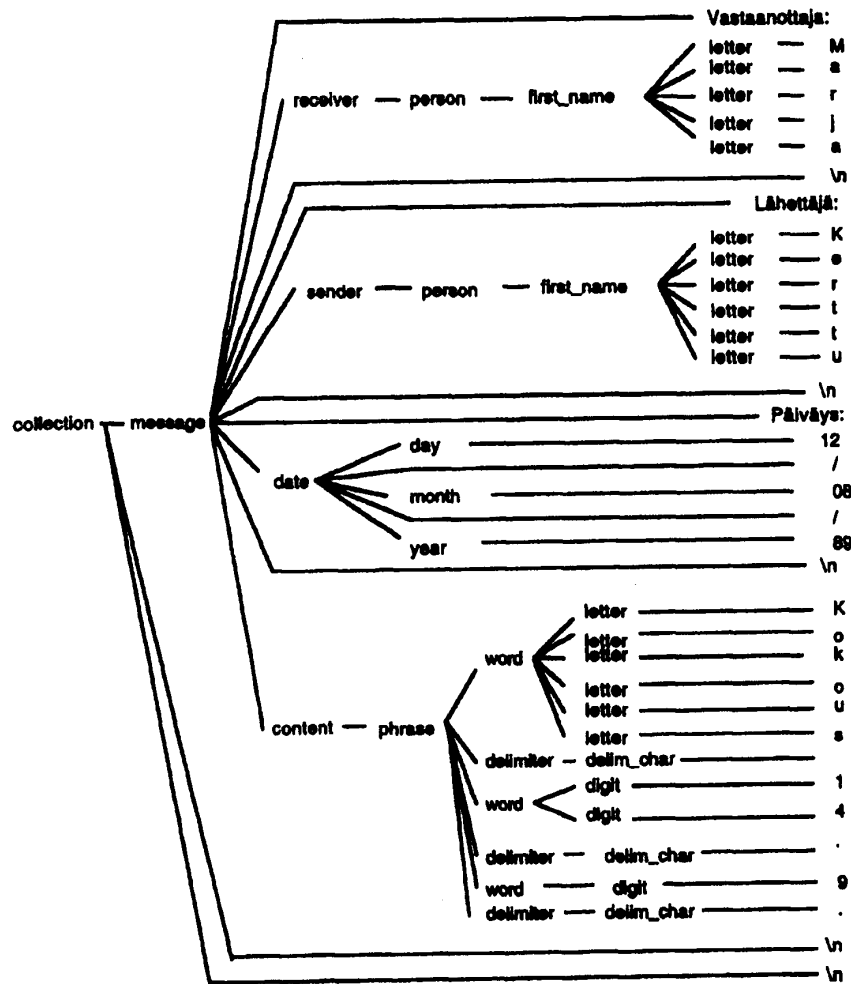


Fig. 3. A parse tree for the grammar in Figure 2.

date "12/08/89", the grammar shows the meaning of the substrings "12", "08", and "89".

A text database whose schema is defined by a grammar can be represented by a parse tree. The parse tree is a hierarchical representation of the text and a graphical representation of a derivation. In the tree, each parent with its children corresponds to an application of a production in the derivation. Figure 3 shows a parse tree using the grammar in Figure 2. The parse tree is a structural description for the first message printed in Figure 1.

78 Alri Salminen

```

      (1)
collection =>
      (2)
message 'n' 'n' =>
      (4)
'Vastaanottaja: ' receiver 'n' 'Lähettiläjä: ' sender 'n' 'Päiväys: ' date 'n' content 'n' 'n' =>
      (6)
'Vastaanottaja: ' person 'n' 'Lähettiläjä: ' sender 'n' 'Päiväys: ' date 'n' content 'n' 'n' =>
      (7)
'Vastaanottaja: ' first_name 'n' 'Lähettiläjä: ' person 'n' 'Päiväys: ' date 'n' content 'n' 'n' =>
      (10)
'Vastaanottaja: ' letter letter letter letter letter 'n' 'Lähettiläjä: ' person 'n'
                        'Päiväys: ' date 'n' content 'n' 'n' =>
'Vastaanottaja: ' 'M' letter letter letter letter 'n' 'Lähettiläjä: ' person 'n'
                        'Päiväys: ' date 'n' content 'n' 'n' =>
      (15)
...
Vastaanottaja: ' 'M' 'a' 'r' 't' 'a' 'n' 'Lähettiläjä: ' 'K' 'e' 'r' 't' 'u' 'n' 'Päiväys: ' ... ' 'n' 'n'

```

Fig. 4. A derivation corresponding to the parse tree in Figure 3.

The nonterminal nodes of the parse tree are labeled by nonterminal symbols, the terminal nodes by terminal symbols. The parse tree corresponds to the derivation shown in Figure 4. In each step of the derivation, the leftmost nonterminal symbol is replaced by a variant of the right side of the corresponding production. The productions used are indicated by their numbers above the symbols \Rightarrow . The derivation begins by the application of the production (1). In the tree, the application is shown as the three children of the root. The derivation ends with the application of production (15) which produces the character '.' to the end of the message.

2.2 Text Types and Text Entities

In text data modeling, we must be able to express the abstractions needed for managing the text entities of an application. This is the role played by the context-free grammar. The text entities defined by a grammar $S = (A, N, P, s)$ are of two kinds: character strings and hierarchical structures. Each nonterminal symbol in N represents a set of character strings, a set of nodes (together with corresponding subtrees) in all possible parse trees for the grammar, and also a set of nodes in a given parse tree. Hence we call the nonterminal symbols in N *text types*. To be able to consider a production with t on its left side as a mechanism for defining a type t , we write the grammar such that each nonterminal symbol appears only once as the left side of a production.

Let $S = (A, N, P, s)$ be a context-free grammar, t a text type in N , and X a parse tree for S . We define the values of type t and a part of type t as follows:

Definition. The *values of type t* (or *t values*) are the terminal symbol strings in the language generated by the grammar $S(t) = (A, N, P, t)$, i.e.,

ACM Transactions on Information Systems, Vol. 13, No. 1, January 1995.

the grammar derived from S by choosing t as the start symbol. In X , a node n labeled by a nonterminal symbol is a *part* if it is not a single child of a parent. The subtree x with n as its root is the *state* of n , and the string produced by concatenating the terminal symbols of x (from left to right) is the *value* of n . The part is a *part of type t* (or a *t part*) if t is the label of n or the label of a node n' in the state of n such that the path from n to n' (including n and n') contains no other parts than n .

This definition shows that not all nodes labeled by nonterminal symbols are parts. A single child of a parent is regarded as renaming a part, not as a separate part itself. As well, if the grammar allows the derivation of an empty string from a nonterminal symbol t then a leaf node may be labeled by t . In such a case, the t part has an empty string as its value. For example, if the first production in Figure 2 is replaced by the production

collection ::= (message '/' n' '/' n')*

there might be a parse tree consisting of one part with an empty value.

Example. Consider the definitions of the types `delimiter` and `delim_char` in the grammar of Figure 2, given by the productions (10) and (15):

(10) `delimiter` ::= `delim_char` +

(15) `delim_char` ::= `'|'|'|'|'$|...`

The values of type `delim_char` are the strings consisting of one character: `"|", "|", "|", "..."`. The values of type `delimiter` consists of nonempty strings of the `delim_char` values. Thus, for example, the string `"|..."` is a value of type `delimiter`.

In the parse tree of Figure 3, the nodes labeled by `collection`, `message`, `receiver`, and `letter` are examples of parts, whereas the nodes labeled by `person` or `first_name` are not. The node labeled by `receiver` is a part of type `receiver` and a part of types `person` and `first_name`. The value of the part is `"Marja"`. Figure 5 shows the parts of the parse tree of Figure 3 such that the state of each part is circled. The terminal symbols like `'Vastaaottaja:'` and `'/'` may carry information to the reader when she or he sees them printed in a message. However, in the database, they are not regarded as meaningful parts. Hence no text types are associated with them.

2.3 Properties

Each of the text types t of grammar can be considered as a logical operation which tests if a part of a parse tree is a part of type t . For example, in Figure 5, the properties `receiver`, `person`, and `first_name` are all true for the part whose value is `"Marja"`, but the property `sender` is not true for the part. On the other hand, there are also other logical operations than just types which can be defined for parts. The logical operations on parts are called *properties*.

There is a group of properties we have defined in Salminen and Tompa [1994] for parts in any parse tree, for any context-free grammar. Figure 6 lists some of them. (P1) indicates that each type t is a property. Other properties are of the form $t(q)$ where q is a constraint for parts of type t .

80 • Airi Salminen

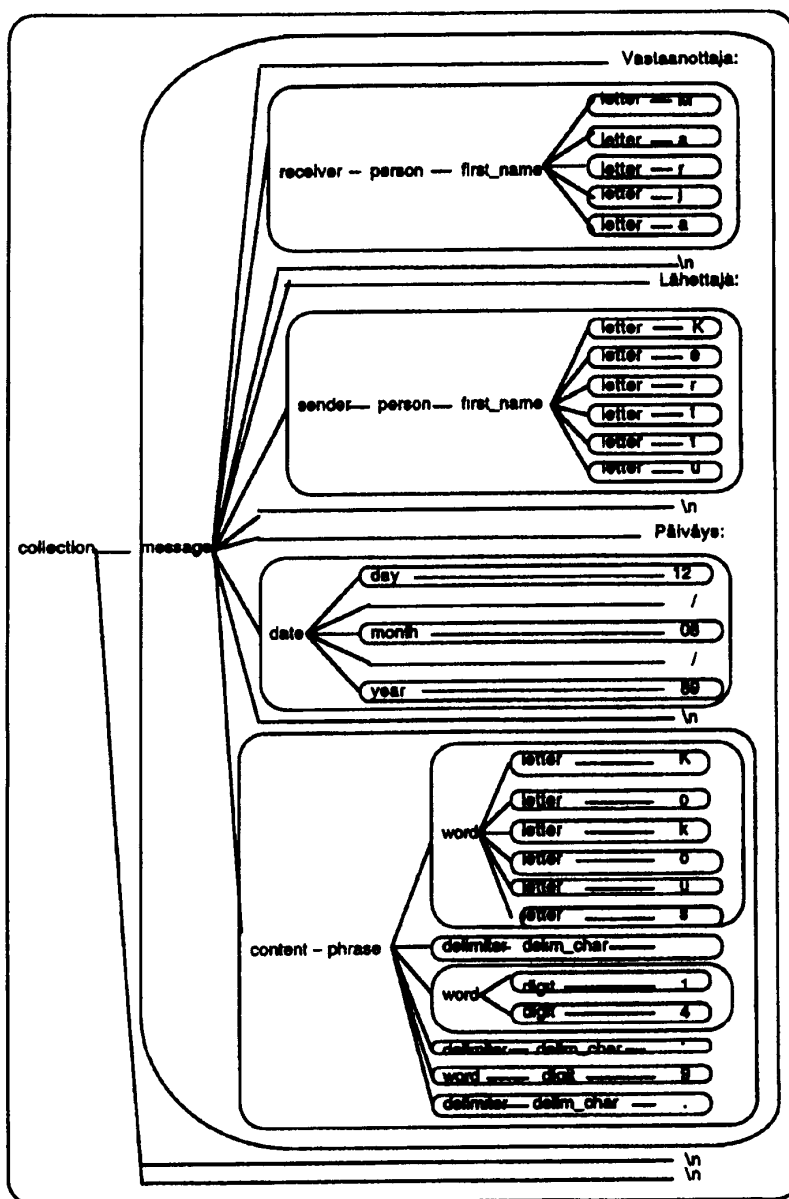


Fig. 5. Parts in the parse tree of Figure 3.

- (P1) t
- (P2) $t\{s\}$
- (P3) $t\{is\ p\}$
- (P4) $t\{in\ p\}$
- (P5) $t\{by\ p\}$
- (P6) $t\{value\ equals\ part\ p\}$
- (P7) $t\{where\ p\}$

Fig. 6. Some universal properties of text parts.

In (P2), s is a character string, and the operation $t\{s\}$ tests if the value of a t part is equal to s . For example, the properties `sender{"Kerttu"}`, `person{"Kerttu"}`, and `first_name{"Kerttu"}` are true for the sender part in Figure 5.

In the recursive properties (P3)–(P7), p is a property. The property $t\{is\ p\}$ tests if a part of type t obeys the property p . The property is convenient for testing if a part of a given type is at the same time also part of another type. For example, in Figure 5, the property `first_name\{is receiver\}` is true for the first name of the receiver, not for the first name of the sender. The properties (P4)–(P6) test the context of a part. The property $t\{in\ p\}$ is true if the argument is a t part and is contained in (the state of) a part obeying the property p . The property $t\{by\ p\}$ tests if the argument is followed by a part for which p is true, in the preorder of parts. For example, the property `sender\{by date{"12/08/89"}\}` is true for the sender part in Figure 5. The property $t\{value\ equals\ part\ p\}$ is true if the value of the argument is equal to the value of another part obeying the property p . For example, the property `word\{value equals part receiver\}` would be true for a word in the content of a message if the word's value was the same as the value of a receiver part. Property (P7) tests parts contained in the argument part. The property $t\{where\ p\}$ is true if the argument contains a part obeying the property p . In testing the property, the context consists of the state of the argument part. The nodes outside the argument have no effect on the result. For example, the property `message\{where word\{value equals part receiver\}\}` is true for a message if it contains a word which is the same as the value of the receiver part in the same message. Receivers outside the message are not tested.

In addition to the common properties defined for any part, special properties for parts defined by a specific schema may be introduced. For example, consider the grammars containing the following productions:

```
digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
number ::= digit +
```

For these grammars we can define the arithmetic comparison properties $t\{= n\}$, $t\{\leq n\}$, $t\{\geq n\}$, $t\{< n\}$, and $t\{> n\}$ where n is a value of type number. The properties test if the value of a part is a number value and if the specified comparison holds between the value and n .

In our model, we use properties in two different ways: to extend grammars

to constrained grammars and to express conditions for data access. In a constrained grammar nonterminal symbols on the right side of productions of a context-free grammar are replaced by properties. Let $G = (A, N, P, s)$ be a context-free grammar and F the set of properties defined for G . Then a constrained grammar is defined as follows:

Definition. A constrained grammar corresponding to G is a 4-tuple (A, N, P', s) where P' is a set of constrained production. Each constrained production $t ::= \alpha'$ in P' is derived from a corresponding production $t ::= \alpha$ of P such that either $\alpha' = \alpha$, or one or more text type occurrences t' of α have been replaced in α' by a property $t'(q)$ of F .

Thus in our constrained grammars, constraints are associated with nonterminal symbols in a similar way as in attribute grammars [Knuth 1968], and we are able to specify data constraints which are difficult or impossible to express by a context-free grammar. An example of a constrained grammar will be given in the following section.

3. DESCRIPTION OF THE CANADIAN PATENT REPORTER AS A SOURCE TEXT

Let us now consider a real and more complicated text database: the Canadian Patent Reporter (C.P.R.). Figure 7 shows a page from the printed form of C.P.R. C.P.R. is published in three series each of which consists of volumes. The first and second series contain 147 volumes together. In the third series 48 volumes were published to the end of 1993.

A volume of C.P.R. consists of court case descriptions. Figure 7 shows the beginning of such a case description. The description is divided into sections. There are seven different types of sections: style of cause, court, key word lists, citations, council names, decision, and summary. The page in Figure 7 shows four whole sections and the beginning of the fifth section. From the top the sections are: style of cause ("COLECO CANADA LTD. v. MUNRO GAMES LTD."), court, key word lists, and summary. The text after the bar belongs to the decision section. The style-of-cause section shows the plaintiff and defendant of the case. A key word list includes key word phrases associated with the case. They are chosen by the publisher of C.P.R. and are attempts to summarize or paraphrase the material in the decision or summary sections. In 1989, a classification for providing consistent categorization of legal topics was published, and thereafter at least the first key word phrase of a key word list has been chosen from the classification scheme. The decision and summary sections consist of natural language text, divided into paragraphs. The text of the paragraphs may include references to other cases in C.P.R. For example, in Figure 7 the string "25 C.P.R. (2d) 126" refers to page 126 of volume 25 in the second series.

In the hypertext retrieval system for C.P.R. we use as the source text the electronic form in which some text parts are denoted by tags. The text of the printed C.P.R. contains figures; in the electronic form, a figure is replaced by a paragraph whose text indicates that in the published volume there is a figure. Figure 8 shows a schema for the electronic form of C.P.R. The symbols

176

CANADIAN PATENT REPORTER

59 C.P.R. (2d)

COLECO CANADA LTD. V. MUNRO GAMES LTD.

*G. Partington, Acting Chairman of the Trade Marks Opposition Board.
December 4, 1979.*

Trade marks - Opposition - PRO STARS for games - PRO HOCKEY - PRO FOOTBALL, PRO SOCCER, PRO STARS HOCKEY, PRO STARS BASKETBALL and PRO ACTION SOCCER for games - Prior use - Evidence - Distinctiveness - Date for determination - Similarity in marks - Identity of wares and channels of trade - Marks confusing - Application refused.

The applicant sought to register the trade mark PRO STARS for use in association with games, namely, toy hockey games, basing the application on proposed use. The opponent based its opposition on grounds of entitlement and distinctiveness by reason of its prior use of PRO HOCKEY, PRO FOOTBALL, PRO SOCCER, PRO STARS HOCKEY, PRO STARS BASKETBALL and PRO ACTION SOCCER in association with similar wares. The evidence of the opponent relative to use after the date of filing of the applicant's mark while not relevant to the issue of entitlement is relevant in respect of the issue of distinctiveness. The evidence of the opponent establishes use of the trade marks PRO and PRO STARS in association with the goods in Canada.

Held, the opponent has established its prior use and non-abandonment of the trade mark PRO in association with toy hockey and soccer games but does not establish use of the trade mark PRO STARS until 1980, subsequent to the filing date of the application in issue.

Neither mark possesses a high degree of distinctiveness. The wares of the parties are identical and would travel through the same channels of trade. The marks are similar in sound, appearance and idea suggested.

The application is refused.

[*Re Andres Wines Ltd. and E. & J. Gallo Winery* (1975), 25 C.P.R. (2d) 126, [1976] 2 F.C. 3, 11 N.R. 560, *re'd* to]

IN THE MATTER OF AN OPPOSITION by Coleco (Canada) Ltd. to application No. 322,084 for trade mark PRO STARS filed by Munro Games Limited.

On May 5, 1969, Munro Games Limited filed an application to register the trade mark PRO STARS based upon proposed use of the trade mark in Canada in association with "games, namely, toy hockey games".

The opponent, Coleco (Canada) Ltd. filed a statement of opposition alleging that the applicant is not the person entitled to registration of the trade mark PRO STARS and that the trade mark PRO STARS is not distinctive in view of the prior use in Canada by the opponent, formerly known as Eagle Toys Ltd.,

Fig. 7. A copy of a page of the Canadian Patent Reporter [Canadian Patent Reporter 1942-1998].

84 . Airi Salminen

- (1) CPR ::= case+
- (2) case ::= '@S' case_identifier section+ '@E'
- (3) section ::= encoding_date | style_of_cause | citations | court |
decision | keyword_lists | summary | council_names
- (4) encoding_date ::= '@1' string
- (5) style_of_cause ::= '@2' string
- (6) citations ::= '@3' CPR_citation citation*
- (7) court ::= '@4' string '@D' date
- (8) decision ::= '@5' paragraph +
- (9) keyword_lists ::= '@6' keyword_list ('@P' keyword_list)*
- (10) summary ::= '@7' paragraph+
- (11) council_names ::= '@8' string
- (12) citation ::= ('@H' | '@N') string
- (13) keyword_list ::= string
- (14) CPR_citation ::= string
- (15) paragraph ::= '@P' string
- (16) case_identifier ::= string
- (17) date ::= string

Fig. 8. A schema for the Canadian Patent Reporter

beginning with @ are tags. The tagging of the source text could be done by SGML style as well. In such a case, our schema would correspond to the SGML document type definition of the source.

The grammar in Figure 8 describes the structure of the source text, but it is not detailed enough for developing a parser for the indexing program. A more detailed source text description is given by a constrained grammar, as shown in Figure 9. In this description we anticipate the way the text will be indexed. The property `CPR_citation{not value equals part CPR_citation}` in production (6) indicates that the initial citation in the citations section is the unique CPR citation for the case, i.e., all CPR citations are distinct and can thus be used to identify a case.

For indexing purposes, a key word list consists of key word phrases, separated by dashes, as indicated in production (13). These phrases are chosen by indexers prior to publication of C.P.R. We wish to index individual words in these phrases. Thus, in production (14), we define a key word phrase as consisting of one or more key words separated by delimiters.

In Figure 8, a paragraph was defined as a string beginning with a tag. For indexing, we want to identify from paragraphs the references to other cases and the words which are used as key words in a key word list of the case. To specify these text items to the indexing routine, we need additional informa-

ACM Transactions on Information Systems, Vol. 13, No. 1, January 1995.


```

(1)  CPR      ::= case(where
                    kword(value equals part keyword(in keyword_list))+
(2)  case      ::= 'N' '@S' case_identifier section+ 'N' '@E'
(3)  section   ::= encoding_date | style_of_cause | citations | court |
                    decision | keyword_lists | summary | council_names
(4)  encoding_date ::= 'N' '@1' 'N' string
(5)  style_of_cause ::= 'N' '@2' 'N' plaintiff 'V.' defendant
(6)  citations  ::= 'N' '@3' 'N'
                    CPR_citation(not value equals part CPR_citation) citation*
(7)  court      ::= 'N' '@4' 'N' string 'N' '@D' 'N' date
(8)  decision   ::= 'N' '@5' 'N' paragraph+
(9)  keyword_lists ::= 'N' '@6' 'N' keyword_list ('N' '@P' 'N' keyword_list)*
(10) summary    ::= 'N' '@7' 'N' paragraph+
(11) council_names ::= 'N' '@8' 'N' string
(12) citation    ::= ('N' '@H' | 'N' '@N') 'N' string
(13) keyword_list ::= keyword_phrase ('-' keyword_phrase)* ':'
(14) keyword_phrase ::= keyword (delimiter keyword)*
(15) CPR_citation ::= first_series | second_series | third_series
(16) keyword      ::= (letter | digit)+
(17) paragraph    ::= 'N' '@P' 'N' (reference | kword | string)+
(18) first_series  ::= volume{≤ 65} 'C.P.R. (1d)' page | volume{≤ 65} 'C.P.R.' page
(19) second_series ::= volume{≤ 82} 'C.P.R. (2d)' page
(20) third_series  ::= volume{≤ 30} 'C.P.R. (3d)' page
(21) volume        ::= number{≥ 1}
(22) page          ::= number{≥ 1 and ≤ 350}
(23) delimiter     ::= delim_char (delim_char | stop_word)*
(24) number         ::= digit+
(25) case_identifier ::= string
(26) plaintiff      ::= string
(27) defendant      ::= string
(28) date           ::= string
(29) string         ::= char+
(30) char           ::= delim_char | letter | digit
(31) stop_word      ::= 'a' | 'an' | 'of' | ...
(32) delim_char     ::= ':' | ';' | ',' | 'N' | ...
(33) letter         ::= 'a' | 'b' | ...
(34) digit          ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
(35) reference      ::= first_series(value equals part CPR_citation) |
                    second_series(value equals part CPR_citation) |
                    third_series(value equals part CPR_citation)
(36) kword          ::= keyword

```

Fig. 9. A constrained schema for the Canadian Patent Reporter.

tion, which we give by properties. In the constrained schema, production (17) defines a paragraph as a sequence consisting of items called reference, kword, and string. By a reference part we mean a part whose value is equal to the value of some CPR_citation part in the collection. This is expressed in production (35); a reference part is defined to be a first_series, second_series, or third_series part such that its value equals the value of some CPR_citation part. The text type kword refers to a key word occurrence in a paragraph. In production (36), parts of type kword are defined to be parts of type keyword. On the other hand, the constraint in production (1) indicates that the value of any kword part has to be a string appearing as a key word in a key word list of the same case.

In productions (23), (31), and (32), a delimiter is defined to consist of characters such as space, comma, and so on, or stop words such as 'a', 'an', 'the', 'of', and so on. Once these stop words have been identified in the grammar, we can exclude them from the index terms.

We use the grammar of Figure 9 in the design of the source text parser, for documentation purposes, and to develop the description of the indexed text. The form defined in Figure 9 will not normally be known to end users.

4. DEFINING THE INDEXING

The constrained grammar, as shown in Figure 9, describes the structure of the original text and the associated specific properties of text parts needed for purposes of input control and analysis. The indexing itself is described by grammar transformations and by a specification of the types whose objects are used as elements in the index, i.e., which elements in the text form a basis for querying and browsing. After indexing, the text is usually viewed as being less structured than the original text, and some of the parts or constant strings in the original text are hidden, since they are not used to build the index table. A grammar transformation is given by a pair of grammars—an *input grammar* and *output grammar*—and a general algorithm for transforming a parse tree for the input grammar to a parse tree for the output grammar. Grammar transformations have been used earlier, e.g., Aho and Ullman [1972], Furuta and Stotts [1988], Kilpeläinen et al. [1990], and Pratt [1971]. To describe the relationship of the original source text and the indexed text, two kinds of grammar transformations will be used: the first is used for hiding parts or constants, the second for suppressing. In both cases, the general transformation algorithm is derived from the algorithm of Aho and Ullman [1972] with slight changes. In the following we will describe an indexing specification for C.P.R.

4.1 Hiding Parts

The indexing of C.P.R. is applied to the source text described by Figure 9. Hence, the grammar of Figure 9 is the input grammar of the first transformation. The transformation itself is specified in the upper part of Figure 10 by a set of *transformation productions*. The production numbers in Figure 10 refer to corresponding productions in Figure 9. The transformation hides the tags, the new line characters preceding and following tags, and the parts of types *case_identifier* and *encoder_date*. The deletions of tags and new line characters are accomplished by giving in a new form all productions which earlier had contained tags. The deletion of type *case_identifier* can be seen in production (2): a case is now just a sequence of section items. Type *encoding_date* is deleted by removing it from production (3).

The output grammar of the transformation consists of the transformation productions, and of those productions of the input grammar which have no corresponding transformation production, excluding the useless productions, i.e., productions not needed in any derivation. (Thus, the *case_identifier* and *encoding_date* productions are not included in the output grammar.)

ACM Transactions on Information Systems, Vol 13, No 1, January 1995

```

(2)  case      ::= section+
(3)  section   ::= style_of_cause | citations | court | decision |
                    keyword_lists | summary | council_names
(5)  style_of_cause ::= plaintiff ' V. ' defendant
(6)  citations ::= CPR_citation citation*
(7)  court     ::= string '\n' date
(8)  decision  ::= paragraph+
(9)  keyword_lists ::= keyword_list keyword_list*
(10) summary   ::= paragraph+
(11) council_names ::= string
(12) citation  ::= string
(17) paragraph ::= (reference | kword | string)+

```

```

(6)  citations ::= CPR_citation txt*
(7)  court     ::= txt
(11) council_names ::= txt
(12) citation  ::= txt
(14) keyword_phrase ::= keyword (txt keyword)*
(16) keyword   ::= txt
(17) paragraph ::= (reference | keyword | txt)+
(26) plaintiff  ::= txt
(27) defendant  ::= txt

```

Fig. 10. Productions describing the indexing transformation for the Canadian Patent Reporter.

4.2 Surpressing

The second set of productions in Figure 10 hides some structural details of text but not any character strings. The input grammar of the second transformation is the output grammar of the first transformation. Type *txt* in the transformation productions is a special type used for surpressing, i.e., hiding, parts which are not indexed. Type *txt* is supposed to be defined as

$$\text{txt} ::= (a_1 | \dots | a_k)^+$$

where a_1, \dots, a_k are the terminal symbols of the original grammar. A production of the form $t ::= \text{txt}$ hides the structure of all t parts. For example, production (16) indicates that a key word is viewed as an atomic part instead of a sequence of letters and digits (as was defined earlier). If *txt* is on the right

88 • Ari Salmunen

side of a t -production such that there are also other symbols, then txt is associated with an appearance of a type t' on the right side of the t -production in the input grammar. The production indicates the hiding of the t' parts in t parts. For example, in the transformation production (6), type txt is associated with the type citation. By this production we indicate that type citation will not be visible after indexing, i.e., that only C.P.R. citations, not the remaining citations, will be indexed at that point. In production (17), type kword has been replaced by the more general type keyword . Type kword was introduced for constraining the key word occurrences in paragraphs to the key words appearing in the key word list of the case. After indexing, type kword is not needed any more.

4.3 The Indexing Schema

The output grammar of the second transformation consists again of the transformation productions and of those productions of the input grammar which have no corresponding transformation production (excluding the useless productions). The output grammar is called an *indexing schema*. Figure 11 shows the indexing schema for the Canadian Patent Reporter. Production numbers refer to the corresponding productions in Figure 9 and 10. From the indexing schema we distinguish the text types whose parts are chosen as index elements. The types are called *index types*. In our sample text, the index types are the following:

`style_of_cause`, `plaintiff`, `defendant`, `CPR_citation`,
`keyword_phrase`, `keyword`, `reference`.

The values of the `keyword_phrase` parts are the index phrases chosen by the publisher. A key word phrase of one case is not necessarily used in the same form in any other case, although it might indeed be considered as a content indicator of other cases as well. Therefore, we have also indexed individual words in key word phrases and their appearances in the summary and decision sections for index terms.

Indexing can be considered as a method of choosing from the data of a document collection those pieces which best describe the content of documents or document parts, and of organizing the data so that efficient information retrieval based on content indicators is possible. The chosen content indicators are stored in an index table from which data can be accessed more efficiently than from the documents.

In our model, data access from the indexed text will be based on describing the properties of parts. We suppose that the information needed for testing a property is in the index table. As in Burkowski [1992], we suppose that the index table contains the index elements and their positions, and positions of parts containing index elements. Thus we can combine the structural properties with properties testing the index element values. All document parts which do not contain index elements or which are not contained in index elements are viewed in the indexing schema as atomic parts. These parts will

ACM Transactions on Information Systems, Vol 13, No. 1, January 1995.

- (1) CPR ::= case+
- (2) case ::= section+
- (3) section ::= style_of_cause | citations | court | decision |
keyword_lists | summary | council_names
- (5) style_of_cause ::= plaintiff ' V. ' defendant
- (6) citations ::= CPR_citation txt*
- (7) court ::= txt
- (8) decision ::= paragraph+
- (9) keyword_lists ::= keyword_list+
- (10) summary ::= paragraph+
- (11) council_names ::= txt
- (13) keyword_list ::= keyword_phrase ('-' keyword_phrase) * ''
- (14) keyword_phrase ::= keyword (txt keyword) *
- (15) CPR_citation ::= first_series | second_series | third_series
- (16) keyword ::= txt
- (17) paragraph ::= (reference | keyword | txt) +
- (18) first_series ::= volume ' C.P.R. (1d) ' page | volume ' C.P.R. ' page
- (19) second_series ::= volume ' C.P.R. (2d) ' page
- (20) third_series ::= volume ' C.P.R. (3d) ' page
- (21) volume ::= txt
- (22) page ::= txt
- (26) plaintiff ::= txt
- (27) defendant ::= txt
- (35) reference ::= first_series | second_series | third_series

Fig. 11. Productions of the indexing schema for the Canadian Patent Reporter.

be included in the screen representation of text, but their properties may not be used as retrieval criteria.

4.4 Special Properties for Text Types

To specify a set of index elements for browsing or querying, we can use all of the universal properties. For example, recall that there was a property $t(s)$ for testing if the value of a t part equals the character string s . Thus, for example, the property `keyword_phrase("Similarity in marks")` is true for a part of type `keyword_phrase` if its value is the string "Similarity in marks". In many cases we need fuzzier properties for testing index elements. For such purposes we can define the property $t(\text{matches } s)$ where s is a character string. It can be defined such that it is true for the parts of index types under certain conditions. For example, we might define $t(\text{matches } s)$ such that it is

ACM Transactions on Information Systems, Vol. 13, No. 1, January 1995

true for a part of an index type t if s is a prefix of the value of the part, after both the value and s are normalized in the same way. In the normalization we might, for example, change all uppercase letters to lowercase letters and remove extra spaces. The definition may be different for different index types. Special properties offer means to specify the dynamic linking of parts with semantic similarities [Bernstein 1990; Salton et al. 1994].

5. INDEXED TEXT AS HYPERGRAPH

As we have seen, an indexed text from which information is retrieved can be described as a parse tree for the indexing schema. During information retrieval we consider the parts of the tree to be nodes of a hypergraph [Berge 1973]. A hypertext model based on hypergraphs was first introduced in Tompa [1989]. Transient hypergraphs, to replace or to extend static hypergraphs, were then introduced in Shepherd et al. [1990] and Watters and Shepherd [1990].

5.1 Hypergraphs

Definition. A *hypergraph* is a pair $H = (X, E)$, where X is a set of elements called *nodes*, and E is a family of nonempty subsets of X such that any element x of X is included in some subset E_j of the family E . A subset E_j in E is an *edge*. An edge consisting of two nodes is called a *link*.

If the data items of a database are considered to compose a set of nodes, it is possible to create transient edges by queries to the database. The database together with the transient edges is a transient hypergraph. In addition to transient edges, there may be static edges. We consider a parse tree over an indexing schema as a database and the parts of the parse tree as the data items with which we associate edges. All binary edges are links which can be used for browsing. In hypergraphs, the browsing is supported, not only by links, but also by the notion of chain:

Definition. A *chain* is a sequence l_1, \dots, l_n of links such that in each l_k ($1 < k < n$), one node is common with the preceding link l_{k-1} , and the other node in common with the succeeding link l_{k+1} .

A chain can be created from an edge by ordering the nodes in the edge.

5.2 Database State

We considered the indexed text as a transformation of the original text, where the transformation was defined by grammar transformations. The indexed text is thus a parse tree for the indexing schema. In information retrieval, the parse tree for the indexing schema will be the database:

Definition. Let D be a parse tree for an indexing schema S . D is called a *database* over S . A *database state* of D is defined as a pair (H, C) such that $H = (X, E)$ is a hypergraph, and C is a set of chains in H . The nodes X consist of the parts of D , except for the parts of type txt. (Parts of type txt are excluded since type txt was used for expressing suppression.)

In the hypergraph, the parts are considered as a set, not as a hierarchical structure. The edges of a database state are either static links or transient edges. Static links can be created automatically in two different ways: from the ordered hierarchical text structure or from parts referring to other parts. The most useful hierarchical links seem to be the links connecting successive components of a composed part together. For example, to be able to browse successive sections of a case of C.P.R., we need links among the sections. A part referring to another part is denoted in a constrained schema by a property of the form *t*(value equals part *p*). In the schema for the Canadian Patent Reporter, a value of type *reference* was defined as a value of some *CPR_citation* part. Since the values of *CPR_citation* parts are all distinct, it means that a reference equals the value of exactly one *CPR_citation* part. In a database state, there will be a link (*y*, *y'*) for each reference type part *y*. The part *y'* is the *CPR_citation* part of the case referred to in *y*. The static links can be stored in the index table.

As indicated earlier, the transient edges and chains of a database state are created during the retrieval session by the end user. A database state includes the following transient edges and chains:

- (1) A set of edges called *selections*: a selection consists of index type parts of the database. One of the selections is the *current selection*.
- (2) A chain called the *index chain* expressing an order for the nodes of the current selection: one of the nodes in the index chain is the *current index element*.
- (3) An edge called the *current window*, associated with a type called the *window type*.

The edges of the categories (1)–(3) will now be discussed.

5.2.1 Selections. A selection consists of index type parts and is created by the selection operations (which will be introduced in Section 6.1). Thus we could have for our sample database a selection consisting of keyword parts, or a selection consisting of *style_of_cause* parts and *reference* parts. One of the selections is called the current selection, and it consists of the index elements in which the user is at that moment interested.

5.2.2 Index Chain. A selection itself is a set where no order is defined for the elements. In browsing index elements of the current selection, we want to consider index elements ordered. The ordered selection is called the index chain. A possible order for parts can always be taken from their preorder in the parse tree. This order can be chosen as the default order. Another possible order might be taken from the weight of the index elements, where the weight could be determined by the number of appearances of the index element inside a given part (e.g., in the case). The third useful order is the alphabetic, or numeric, order of the index element values.

5.2.3 Current Window. We suppose that a retrieval system offers a capability to specify, not only the index elements the user is interested in, but also the parts (documents) the reader wants to see at one time. Thus we do not predetermine the structures from which the user has to choose a subset for

ACM Transactions on Information Systems, Vol. 13, No. 1, January 1995.

reading. We consider that flexible capabilities are needed for defining the documents for reading and browsing. For example, the reader of C.P.R. might sometimes want to retrieve and browse paragraphs, another time sections or cases.

What parts will be shown is determined by the user specifying a window type t . In a database state, the current window W is a pair (w, c) such that w is a t part, and c is a part, called the current part, contained in w . The screen shows an external representation of part w . The position of the screen cursor determines the current part. The external representation is given by describing the input of a formatting program (e.g., troff, L^AT_EX, or a simple printing program). Formally, the description can be given by transformation productions; in a similar way as in the indexing specifications. Now the input grammar of the transformation is the indexing schema. A transformation production may be derived from a production of the indexing schema by deleting some terminal or nonterminal symbols, or by adding some new terminal symbols. A general algorithm for the actual transformation may be again derived from the algorithm of Aho and Ullman [1972]. For example, if the window type is the type case of the indexing schema for C.P.R., the window representation could be expressed by the following productions:

```
case ::= section ( '/' n '/' n ' section ) *
section ::= style_of_cause | citations | court
style_of_cause ::= ' PLAINTIFF: ' plaintiff ' \ n ' ' DEFENDANT: ' defendant
citations ::= ' CITATION: ' CPR_citation
court ::= ' COURT: ' txt
```

The productions indicate that the sections of a case are separated from each other by two new line characters. From all sections of a case, only the style-of-cause, citations, and court sections are taken to the window representation. In a style-of-cause section, the plaintiff is preceded by the text "PLAINTIFF: ", and the defendant is written on the following line, after the text "DEFENDANT: ". From citations, only the C.P.R. citation is shown and preceded by the text "CITATION: ". The court section is preceded by the text "COURT: ". Figure 12 shows how the case of Figure 7 would be displayed by the transformation productions.

In our prototype system, the window representation for the window types available for the end user is predefined. The capability to specify window representations should also be given to the end users; however, the way in which the end user could express the specification is an area for our future studies.

6. DATA ACCESS FROM INDEXED TEXT

The data access operations on indexed text databases are divided into selection operations, chain operations, and navigating operations. Each of the operations changes the database state. Selection operations update the edge called the current selection; chain operations update the index chain. Navigating operations update the current window, current path, and current

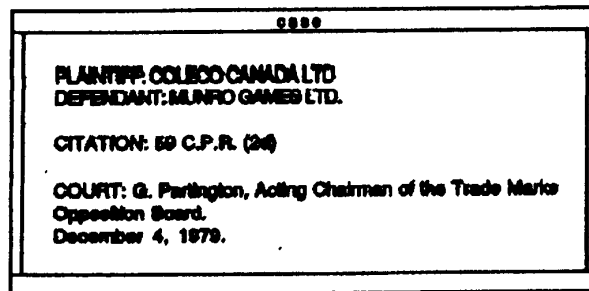


Fig. 12. A representation of a case in a window.

index. Most of the operations we will describe can be expressed by the following edge manipulation operations, introduced in the transient hypergraph-based model [Watters and Shepherd 1990]:

- GETFROM is a unary operation that creates a new edge from an existing edge, according to some condition.
- UNION, INTERSECTION, and DIFFERENCE are binary set operations on edges.
- LINEAR is either a binary or unary operation which creates a set of new binary edges, i.e., links. The operation may have a condition to qualify the resulting edges.

6.1 Selection Operations

A selection operation creates a new current selection. Selections have names by which they can be denoted in selection operations. Selection operations are expressed by the GETFROM, UNION, DIFFERENCE, and INTERSECTION operations. Given a property p , the operation GETFROM chooses from all database parts those for which the property p is true. Properties testing values of parts which are not index elements are not allowed as the selection condition p . This restriction is made because the truth value of the selection conditions will be determined from the data in the index table, not from the data in the original documents.

Suppose the database is defined by the indexing schema in Figure 11. The following are examples of conditions we can write in the GETFROM operation:

- (a) `style_of_cause`
- (b) `style_of_cause{in case{where CPR_citation{is third_series}}}`
- (c) `keyword_phrase{"Similarly in marks"}`
- (d) `keyword{fby keyword{"Company"}}`
- (e) `keyword{in summary}`
- (f) `reference{is first_series}.`

Condition (a) chooses all style-of-cause sections in the database while (b) selects only the style-of-cause sections from the third series of the C.P.R.

ACM Transactions on Information Systems, Vol. 13, No. 1, January 1995

Condition (c) specifies the key word phrases "Similarity in marks"; they occur in a key word list. In (d) and (e), key word occurrences are specified. In general, they occur either in a key word list or in a decision or summary section. In (d), those key word occurrences are chosen which are followed by the key word "Company". In (e), the chosen key words appear in a summary section. Condition (f) specifies the references to the cases in the first series.

In the UNION, INTERSECTION, and DIFFERENCE operations, the arguments are either selections or the edge consisting of all database parts.

6.2 Chain Operations

A chain operation sorts the index elements of the current selection and thus creates a chain. The operation contains one or more criteria for sorting the elements. A sort criterion is either the position of index elements in text, their content, i.e., the alphabetic order of the character strings, or the retrieval weight. These criteria can also be combined together.

6.3 Navigating Operations

Navigating or browsing operations update the current index element, current part, and current window. The current part is specified by the position of the screen cursor, which the reader can move, e.g., by a mouse. Any cursor movement in the window is a browsing operation. Each browsing operation is associated, at least implicitly, with a window type t . The window type determines the type of the current window after the browsing operation. The window types available to a reader are types which contain index elements. The reader chooses the window type in a browsing operation, or there is some default window type. In the Canadian Patent Reporter system, the default window type is case.

All static links can be used for browsing. If the current part (of the current window) is a part n , and there is a link (n, n') in the database state, we can have an operation for moving the window cursor to the part n' . The operation is associated with a window type t . If t is not given by the user, there is a default window type t .

For browsing the index chain, there are operations for moving to the beginning or end of the chain, and forward and backward in the chain. Index chain browsing operations can also include qualification for the index elements.

7. THE IMPLEMENTATION

The cases of the Canadian Patent Reporter are stored as text files. The software used to permit hypertext access consists of three components: the indexing component, the user interface component, and the retrieval component. The indexing component is similar to a programming language compiler. In the compiler, the text is analyzed by a parser. The C.P.R. parser uses basically the grammar shown in Figure 9. Earlier, we described indexing as text transformation. In the compiler, the transformation is implemented by analyzing the text by the original schema, and by creating indexes for the

ACM Transactions on Information Systems, Vol. 13, No. 1, January 1995.

Canada Patent Reporter - Hypertext Interface				
Case	chain	Thesaurus	Mark	Window Operation ?
29 C.P.R.(2d) 1, REGINA V. CANADIAN GENERAL ELECTRIC CO. LTD. et.al.				
Ontario High Court of Justice, Parnell, J.				4
Decision Date: September 2, 1976.				
Keywords: Combines -- Conspiracy -- Agreement amongst three major manufacturers of large lamps -- Manufacturers adopting similar sales plans -- Uniform price lists -- Identical tenders -- Policy of enforcing operation of sales plans -- Agents of manufacturers of consignment -- Vertical conspiracies -- Combines Investigation Act (Can.), s. 32(1)(c) -- Communication among accused -- Pricing				↓
String _____			<clear>	<Again>
<new chain>	<chain Outline>	<Prev link>	<Next link>	<?>
<custom chain>	<case outline>	<prev case>	<next case>	
Current chain: Combines			Link 1 of 100	

Fig. 13. A case window showing user menus in the Canadian Patent Reporter database.

index elements in the text. The indexed text is represented by the indexes together with the text files.

The index of an index type t is a table containing the occurrence positions for each distinct t value. A position is given by the file, section, line, and character position, where the file is identified by its name, and a section by its number within a case. Since the sections are divided into lines, line numbers and character positions in lines are used to denote the precise position of an occurrence inside a section.

The user interface component is implemented by means of *browsers*. A browser determines a window type and a view by which the text is displayed in the window. The default window type is *case*, and the default view to this type shows all parts of a case. In Figure 13, a reader is using the *case browser*. In the *case outline browser*, the view shows an overview of a case only. There is also a *case history browser* providing the user with an overview of the cases and case components visited during the current session. The *selection history browser* lets the user see which selections have been created in the current session. The *index browser* permits the user to examine entries in any of the seven indexes (i.e., the indexes for the text types *style_of_cause*, *plaintiff*, *defendant*, *CPR_citation*, *keyword_phrase*, *keyword*, and *reference*).

The retrieval component is very portable because it makes no assumptions about how the system will eventually interact with the user. The top level of the retrieval component consists of one module for each browser in the

system. These modules export functions for manipulating the current line in the browser. Only a small and well-defined section of the code will have to be rewritten to port the system to different architectures.

8. CONCLUSION

In many environments, the stored data consist of text files which have been created by different systems: text editors, scanning programs, publishing tools, EDI tools. These systems are primarily designed for text manipulation, text formatting, and text transfer purposes, not for managing the text as a whole in a large document collection. Hypertext systems, on the other hand, are designed for authoring and reading text in document collections. For reading, these systems have powerful navigation capabilities but very limited query capabilities [Halasz 1988].

In this article, we introduced a model that supports the creation of document databases for specific application areas. The documents may originate from different systems, as far as their structure is sufficiently consistent. The model permits the information to be accessed by both querying and navigation. The hypertext links are created dynamically from the indexed text. The hierarchical structure of documents is defined by a constrained context-free grammar, and some document parts are specified as index elements. The indexing means transforming the original hierarchical structure to a new hierarchical structure. In information retrieval, parts of the hierarchical structure are considered as nodes of a hypergraph. Hierarchical relationships and references in the text can be defined as static binary edges (links) of the hypergraph. In a query, the reader extends the hypergraph with a transient edge. By ordering the nodes of an edge, the reader creates a transient chain, consisting of a sequence of links. For navigating in the text, the reader uses either static links or links in the transient chains.

Formal grammars have traditionally been used for specifying programming languages. When documents are programs, their correctness is considered very important. In the case of documents written for human readers, we are used to relying on the understanding of the reader. Small differences in the way we write a piece of text do not cause problems when the human reader sees the text. However, when information is retrieved from a large collection of documents, the person needing the information cannot see the documents. If the system contains the capability of retrieving data by a condition, the reader uses her or his knowledge concerning the way text is written when she or he writes the condition. If the knowledge does not correspond to the actual way the text is written, the result of the retrieval may be misleading. Our model leads to the idea of correctness checking for text databases similar to those for program texts, via a constrained grammar.

In this article we did not study problems in the update of documents. Basically, the update problem is similar to the update problem of a program collection consisting of subroutines interrelated to each other. In both cases, a piece of new text may be created by any system. Before the text can be inserted to the database, it is parsed; its correctness is checked; and its

relationship to other text of the database is checked. Similarly, before deleting any piece of text, its relationship to the rest of the text has to be checked.

In our example database system, the documents are legal case descriptions, and the system is intended for retrieving information from them. In the near future, we will be studying user paths in the system in order to evaluate the utility to the user of various types of links and chains. The development of a more general system, where the structure of documents can be defined by the user, is another area of future studies.

ACKNOWLEDGMENTS

The authors gratefully acknowledge David Ellis, Kalervo Järvelin, Jukka Paakki, Carolyn Watters, Peter Willet, and the anonymous referees for their reading and commenting of earlier versions of this article.

REFERENCES

- AHO, A. V. AND ULLMAN J. D. 1972. *The Theory of Parsing, Translation, and Compiling*. Vol. 1 and 2. Prentice-Hall, Englewood Cliffs, N.J.
- APPELT, W. 1991. *Document Architecture in Open Systems. The ODA Standard*. Springer-Verlag, New York.
- BERGE, C. 1978. *Graphs and Hypergraphs*. North-Holland, Amsterdam.
- BERNSTEIN, M. 1990. An apprentice that discovers hypertext links. In *Proceedings of the European Conference on Hypertext*. Cambridge University Press, Cambridge, U.K.
- BLAIR, D. C. AND MARON, M. E. 1985. An evaluation of retrieval effectiveness for a full-text document retrieval system. *Commun. ACM* 28, 3 (Mar.), 289-299.
- BURKOWSKI, F. J. 1992. An algebra for hierarchically organized text-dominated databases. *Inf. Process. Manage.* 28, 3, 333-348.
- CONKLIN, J. 1987. Hypertext: An introduction and survey. *Computer* 20, 9 (Sept.), 17-41.
- DEROSE, S. J. 1990. DynaText: Electronic book indexer/browser. *EPSIG News* 3/4 (Dec.), 1-2.
- DISA. 1990. *ASC X12S/90-116, An Introduction to Electronic Data Interchange*. Data Interchange Standards Assoc., (DISA).
- EGAN, D. E., REMDE, J. R., GOMEZ, L. M., LANDAUER, T. K., EBERHARDT, J., AND LOCHBAUM, C. C. 1989. Formative design-evaluation of "SuperBook." *ACM Trans. Inf. Syst.* 7, 1 (Jan.), 30-57.
- FALOUTSOS, C., LEE, R., PLAIBANT, C., AND SHNEIDERMAN, B. 1990. Incorporating string search in a hypertext system: User interface and signature file design issues. *Hypermedia* 2, 3, 183-200.
- FAWCETT, H. 1989. *PAT Installation Guide*. Centre for the New Oxford English Dictionary, University of Waterloo, Ontario, Canada.
- FURUTA, R. AND STOTTS, P. D. 1988. Specifying structured document transformations. In *Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography*. Cambridge University Press, Cambridge, U.K., 109-120.
- FURUTA, R., PLAIBANT, C., AND SHNEIDERMAN, B. 1989a. A spectrum of automatic hypertext constructions. *Hypermedia* 1, 2, 179-195.
- FURUTA, R., PLAIBANT, C., AND SHNEIDERMAN, B. 1989b. Automatically transforming regularly structured linear documents into Hypertext. *Elec. Pub.* 2, 4 (Dec.), 211-229.
- GLUSHKO, R. J. 1989. Transforming text into hypertext for a compact disc encyclopedia. In *Proceedings of CHI '89*. ACM, New York, 293-298.
- GOLDFARB, C. F. 1990. *The SGML Handbook*, Y. Rubinsky, Ed. Oxford University Press, New York.

ACM Transactions on Information Systems, Vol. 13, No. 1, January 1995

MS 062457

- GONNET, G. H. 1987. Examples of PAT* applied to the Oxford English Dictionary. UW Centre for the New Oxford English Dictionary.
- GONNET, G. H. 1983. Unstructured data bases, or very efficient text searching. In *Proceedings of the 2nd SIGACT-SIGMOD Symposium on Principles of Database Systems*. ACM, New York, 117-124.
- GONNET, G. H AND TOMPA, F. W. 1987. Mind your grammar: A new approach to modelling text. In *Proceedings of the 13th Conference on Very Large Data Bases*. VLDB Endowment, 339-346.
- GYBSENS, M., PARADAENS, J., AND VAN GUCHT, D. 1989. A grammar-based approach towards unifying hierarchical data models. In *Proceedings of the 1989 ACM SIGMOD International Conference*. SIGMOD Rec. 18, 2, 263-272.
- HALABZ, F. G. 1988. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems *Commun. ACM* 31, 7 (July), 836-852.
- ISO. 1988. *ISO 9735 Electronic Data Interchange for Administration, Commerce, and Transport (EDIFACT)—Application level syntax rules* 1st ed. Int. Standards Organization, New York.
- KILPILAINEN, P., LINDÉN, G., MANNILA, H., AND NIKUNEN, E. 1990. A structured document database system. In *Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography*. Cambridge University Press, Cambridge, U.K., 139-151.
- KNUTH, D. E. 1968. Semantics of context-free languages. *Math. Syst. Theor.* 2, 2, 127-145.
- KUIKKA, E. AND PENTTONEN, M. 1993. Transformation of structured documents with the use of grammar. *Elec. Pub. Orig. Dissem. Des.* 6, 4 (Dec.), 373-383.
- MACLEOD, I. A. 1991. A query language for retrieving information from hierarchic text structures. *Comput. J.* 34, 3 (June), 254-264.
- MACLEOD, I. A. 1990. Storage and retrieval of structured documents. *Inf. Process. Manage.* 26, 2, 197-208.
- MARCHIONINI, G. AND SHNEIDERMAN, B. 1988. Finding facts vs. browsing knowledge in hypertext systems. *Computer* 21, 1 (Jan.), 70-80.
- PRAATT, T. W. 1971. Pair grammars, graph languages and string-to-graph translations. *J. Comput. Syst. Sci.* 6.
- RAYMOND D. R. AND TOMPA, F. W. 1988. Hypertext and the Oxford English Dictionary. *Commun. ACM* 31, 7 (July), 871-879.
- RITCHIE, I. 1989. HYPERTEXT—Moving towards large volumes. *Comput. J.* 32, 6 (Dec.), 516-523.
- SALMINEN, A. AND TOMPA, F. W. 1994. Data modelling with grammars. Tech. Rep., Dept. of Computer Science, Univ. of Waterloo, Ontario, Canada. To be published.
- SALMINEN, A. AND WATTERS, C. 1992. A two-level structure for textual databases to support hypertext access. *J. Am. Soc. Inf. Sci.* 43, 6 (July), 432-447.
- SALTON, G. 1988. Automatic text indexing using complex identifiers. In *Proceedings of the ACM Conference on Document Processing Systems*. ACM, New York, 135-144.
- SALTON, G. 1981. A blueprint for automatic indexing. *ACM SIGIR Forum* 16, 2 (Fall), 22-38.
- SALTON, G. AND MCGILL, M. J. 1989. *Automatic Text Processing*. Addison-Wesley, Reading, Mass.
- SALTON, G., ALLAN, J., AND BUCKLEY, C. 1994. Automatic structuring and retrieval of large text files. *Commun. ACM* 37, 2 (Feb.), 97-108.
- SHEPHERD, M. A., WATTERS, C. R., AND CAI, Y. 1990. Transient hypergraphs for citation networks. *Inf. Process. Manage.* 26, 3, 395-412.
- STOTTS, P. D. AND FURUTA, R. 1990. Hierarchy, composition, scripting languages, and translators for structured hypertext. In *Proceedings of the European Conference on Hypertext*. Cambridge University Press, Cambridge, U.K., 180-193.
- TAGUE, J., SALMINEN, A., AND MCCLELLAN, C. 1991. Complete model for information retrieval systems. In *Proceedings of the 14th Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, 14-20.

- TOMPA, F. W. 1989. A data model for flexible hypertext database systems. *ACM Trans. Inf. Syst.* 7, 1 (Jan.), 85-100.
- TSICHRITZIS, D. AND LOCHOVSKY, F. H. 1982. *Data Models*. Prentice-Hall, Englewood Cliffs, N.J.
- WATERS, C. R. AND SHEPHERD, M. A. 1990. Transient hypergraph-based model for data access. *ACM Trans. Inf. Syst.* 8, 2 (Apr.), 77-102.
- WILSON, E. 1990. Links and structures in hypertext databases for law. In *Proceedings of the European Conference on Hypertext*. Cambridge University Press, Cambridge, U.K., 194-211.

Received August 1992; revised January 1994; accepted April 1994

Exhibit 25



US005815830A

United States Patent [19]
Anthony

[11] **Patent Number:** **5,815,830**
 [45] **Date of Patent:** **Sep. 29, 1998**

[54] **AUTOMATIC GENERATION OF
 HYPERTEXT LINKS TO MULTIMEDIA
 TOPIC OBJECTS**

5,603,025 2/1997 Tabb et al. 395/602
 5,608,900 3/1997 Dockter et al. 395/613

FOREIGN PATENT DOCUMENTS

0316957 5/1989 European Pat. Off. .
 0501770 9/1992 European Pat. Off. .
 0509947 10/1992 European Pat. Off. .

[76] **Inventor:** **Andre Charles Anthony**, Hunters
 House, Cheshunt/Herts, England, EN7
 6HQ

Primary Examiner—Paul R. Lintz
Attorney, Agent, or Firm—Dick and Harris

[21] **Appl. No.:** **575,728**

[22] **Filed:** **Dec. 18, 1995**

[30] **Foreign Application Priority Data**

Dec. 23, 1994 [GB] United Kingdom 9426165

[51] **Int. Cl.⁶** **G06F 17/30**

[52] **U.S. Cl.** **707/6; 707/1; 707/501;
 707/513**

[58] **Field of Search** **395/601, 606,
 395/762, 774, 326**

[56] **References Cited**

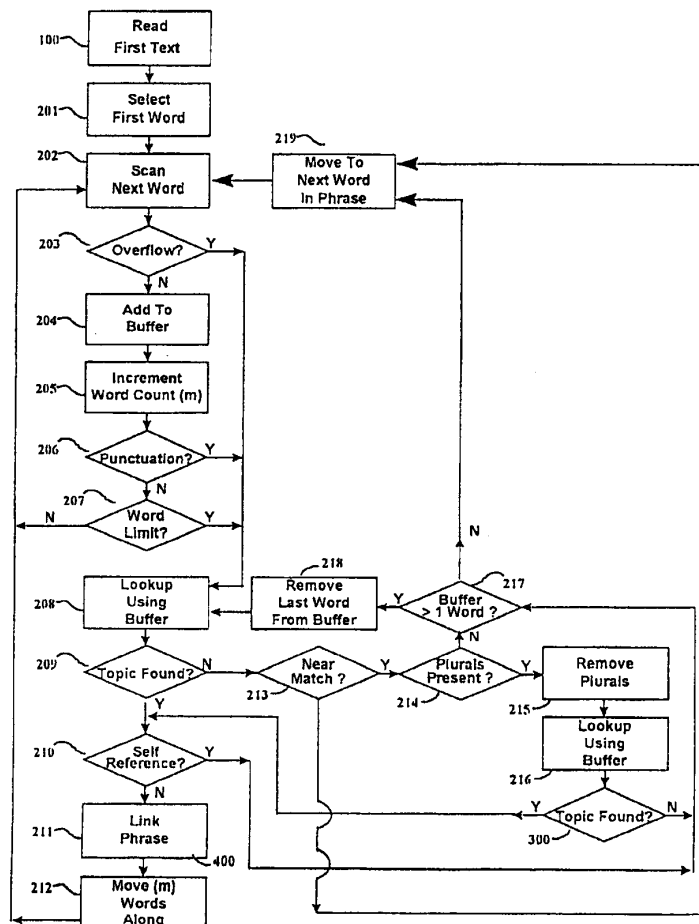
U.S. PATENT DOCUMENTS

4,982,344 1/1991 Jordan 345/346

[57] **ABSTRACT**

Topic objects are stored with textual data objects containing references to other topic objects. The textual data objects are string-correlated to the topic objects to determine which topic objects are referenced in each textual data object. Hypertext links are generated for each reference in the textual data objects. The string-correlation is performed in a buffer in which each portion of text relating to a particular topic, having a unique topic name embedded, is concatenated and matched with the topic names. In the case of no match, one word is removed from the concatenated string to form a new string and the string-correlation is repeated.

25 Claims, 5 Drawing Sheets



MS 067393

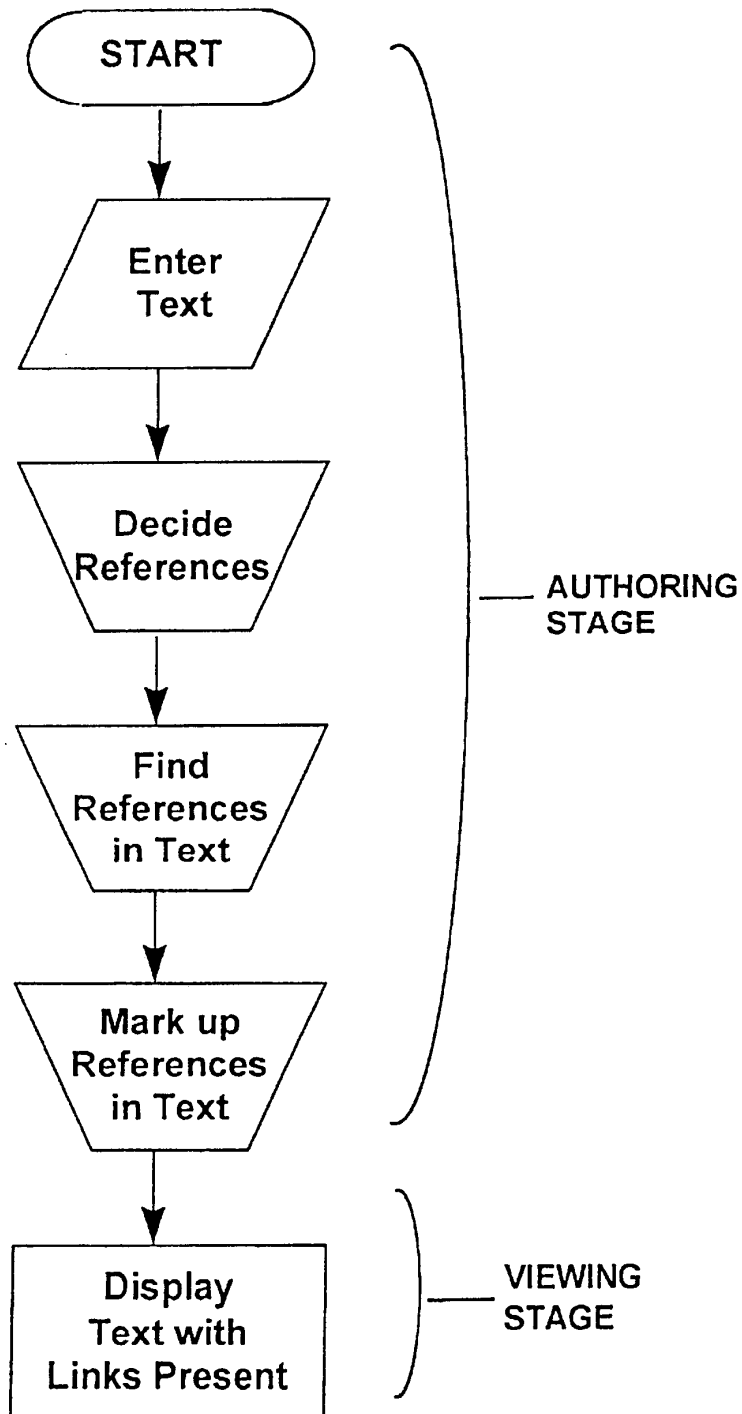
U.S. Patent

Sep. 29, 1998

Sheet 1 of 5

5,815,830

Figure 1



MS 067394

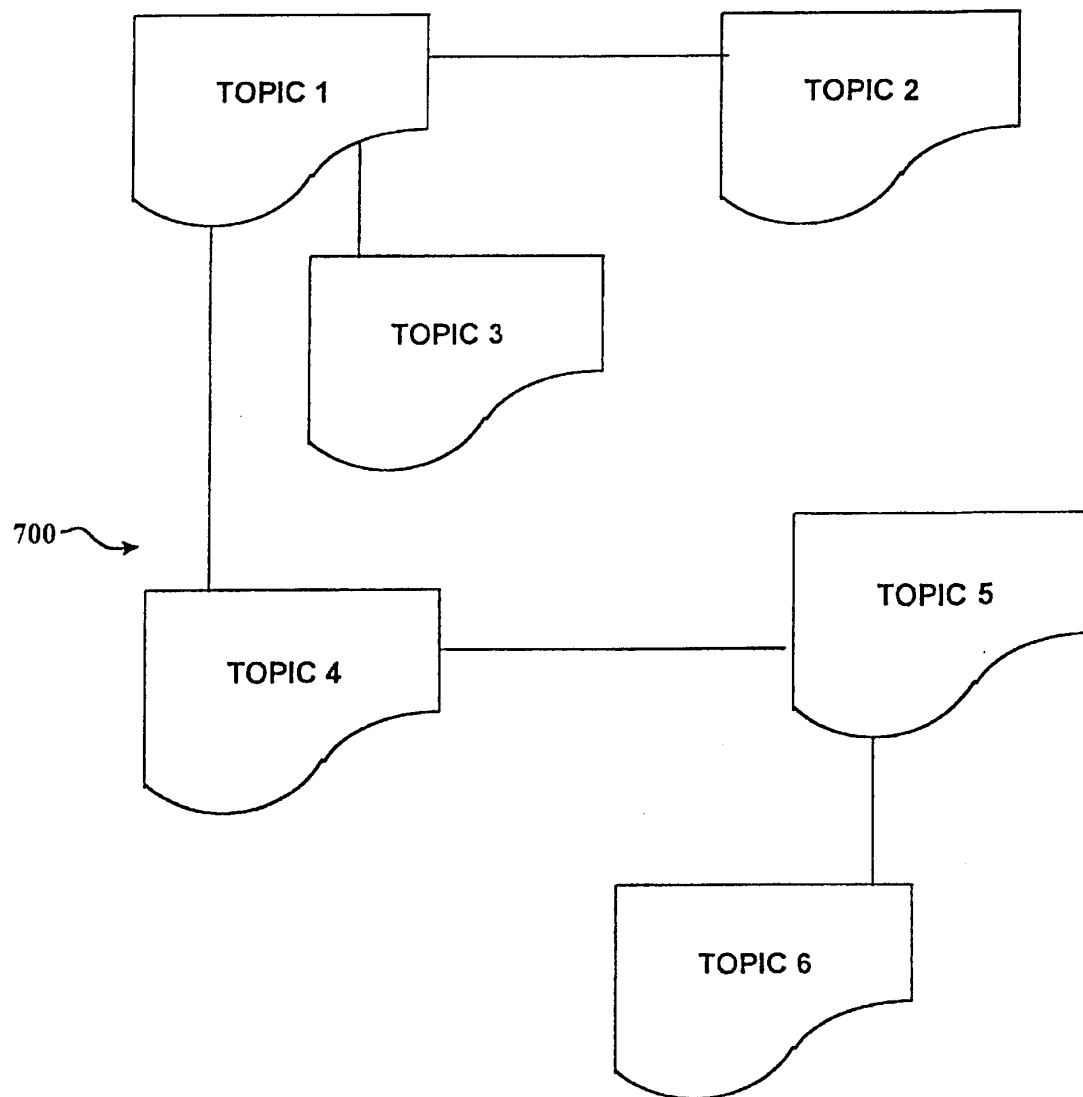
U.S. Patent

Sep. 29, 1998

Sheet 2 of 5

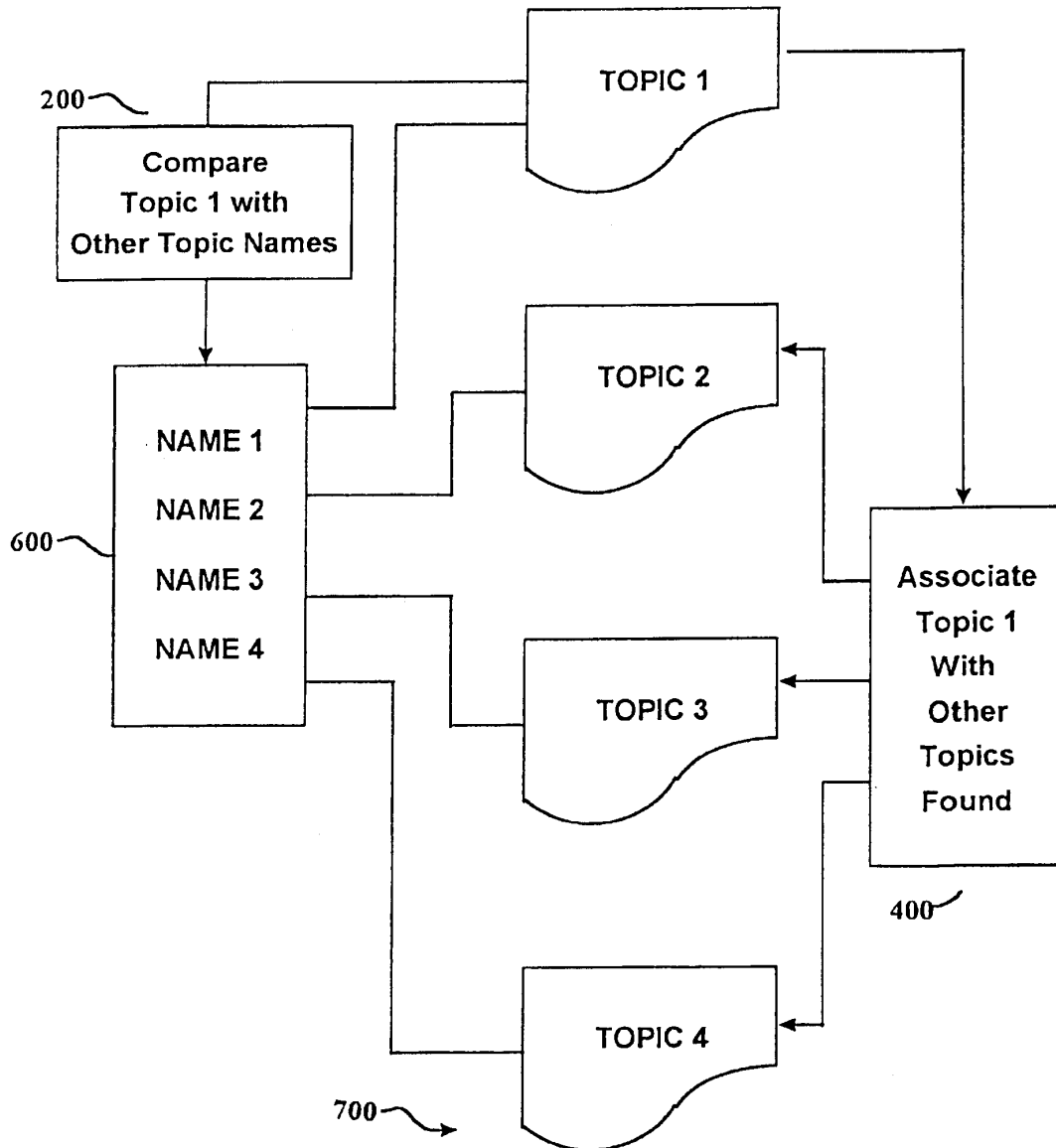
5,815,830

Figure 2



MS 067395

Figure 3

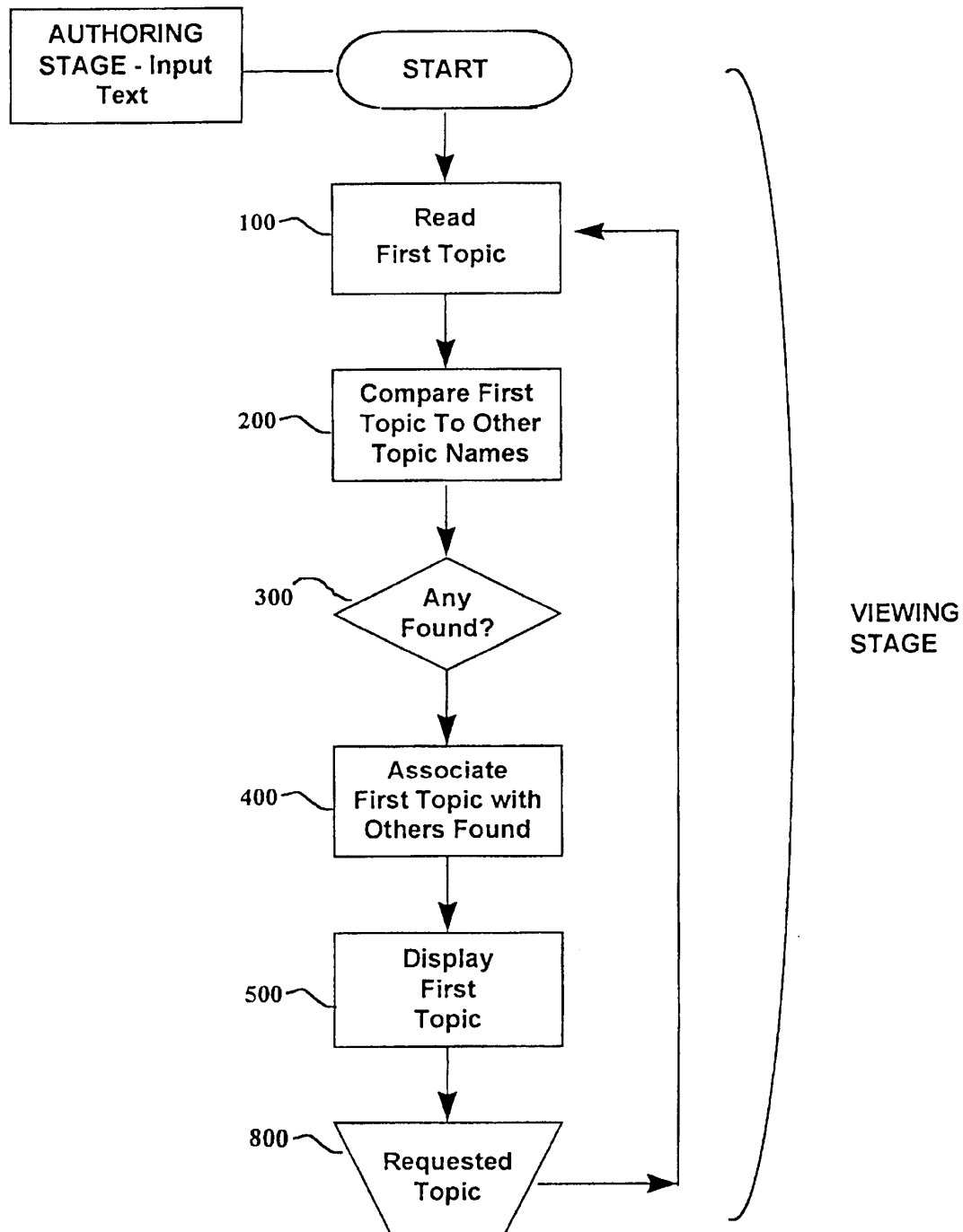


U.S. Patent

Sep. 29, 1998

Sheet 4 of 5

5,815,830

Figure 4

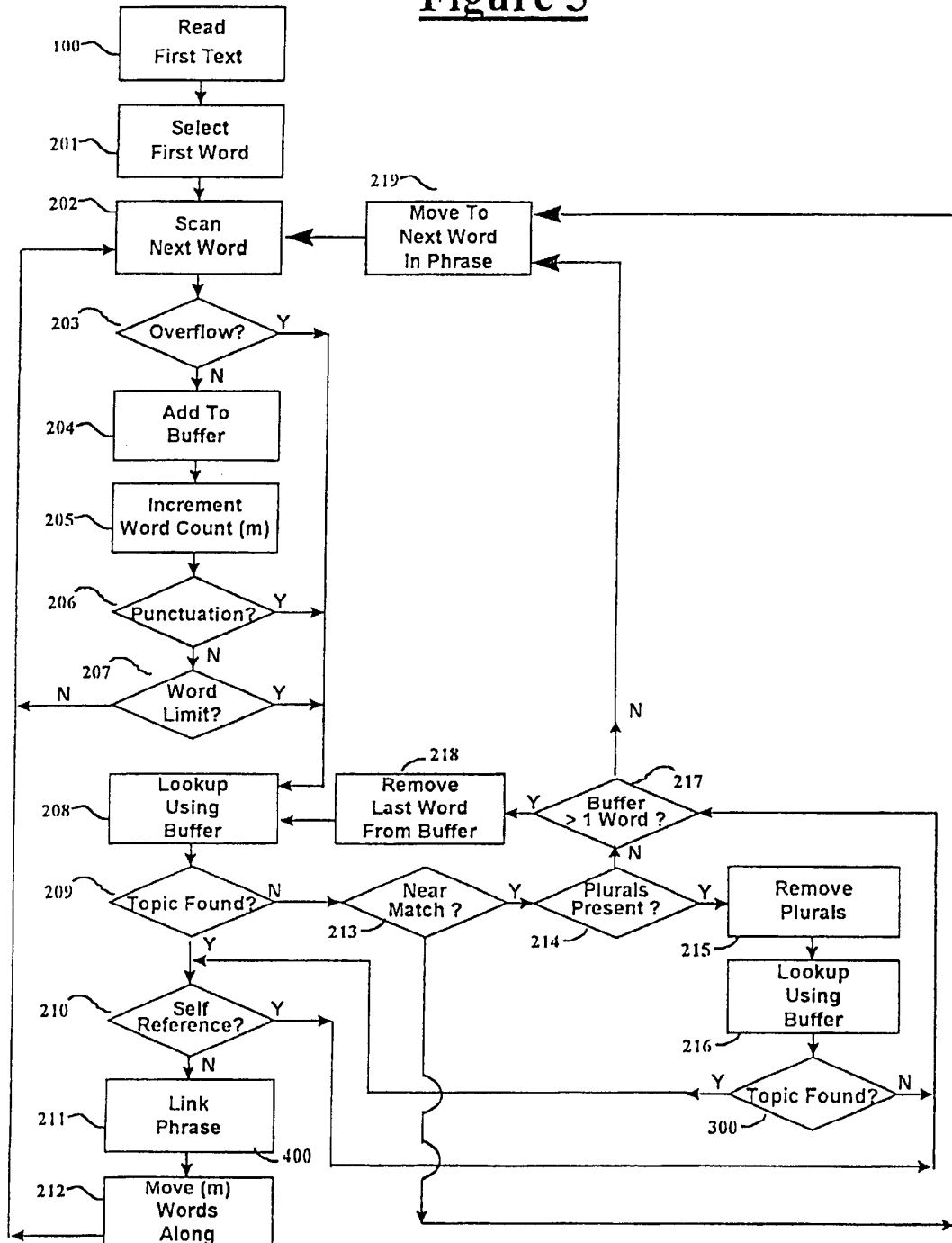
MS 067397

U.S. Patent

Sep. 29, 1998

Sheet 5 of 5

5,815,830

Figure 5

MS 067398

5,815,830

1

AUTOMATIC GENERATION OF HYPERTEXT LINKS TO MULTIMEDIA TOPIC OBJECTS

FIELD OF THE INVENTION

The present invention relates to methods and systems of information management, and, more particularly to Hypertext information retrieval and display.

BACKGROUND OF THE INVENTION

In information management, there is an ever increasing need for systems which provide easy access to information, with minimum time spent in ordering or retrieving the information. In computing, several systems, such as databases and Hypertext, are known for facilitating the organization of information.

A computer database typically comprises a number of records which are grouped in a single file. Each record comprises information stored as a set of fields; each record having the same field types. Proprietary software applications are available for creating and manipulating databases, such as: DBase™, Paradox™, and Approach™ which are available to run on IBM™ compatible machines running DOS or Windows™. Such applications allow data in a database to be rearranged, sorted, presented and printed according to criteria set by the user. The criteria may be, for example, to present a particular subset of records, or to reorder all records in the database. Prior art database manipulation techniques are thus useful for rearranging data which is easily categorisable into different fields. However, such techniques do not allow large quantities of textual data to be presented for easy digestion by the user. Moreover, manual manipulation of the data is necessary, which requires knowledge of the structure of the database.

A second, rather different method for managing information, is to cross-reference documents. Textual documents are more easily digested by cross-referencing than by re-ordering with database applications, as portions of text are cross-referenced to other relevant portions of text, allowing fast and easy access to relevant information. Cross-referencing implemented in this way is termed "Hypertext cross-referencing" or "Hypertext linking". Systems providing such capabilities are called "Hypertext systems". In order for existing hypertext systems to carry out this type of navigation, codes and/or instructions need to be embedded in the text by the author using a given "authoring" program. When the text is displayed using a compatible "viewing" program the codes and instructions, as embedded by the author (but now hidden from view), are interpreted and carried out. A large amount of time and effort is usually expended by the author.

Typically the following steps, as shown in FIG. 1, are carried out during the authoring process:

- (a) identify a word or phrase from which to cross-reference
- (b) identify all occurrences of this word or phrase in the entire text
- (c) determine the location of the referred-to text i.e. its page number or other positional or relative location or its identifying label
- (d) mark each occurrence from step b) to show it is a cross-reference and also attach to it the result found in step c) to tell the system where to navigate to
- (e) repeat steps a to d for all cross-references.

This procedure would usually involve the author in having to mark cross-reference words and phrases with special

2

codes or with some form of computer language. A known standard for coding such links is Hypertext Markup Language (HTML).

Depending on the system used, software support can be found to assist in the above processes, but the author is still required to identify and code cross-references. The coded text is then compiled: which means codes and instructions are validated, navigation links are resolved, text may be compressed. Compilation can take many hours; authoring, in the mechanistic sense, can take many months.

When new text is added to the publication, occurrences of words and phrases are inevitably added which are the same as existing cross-references, the above steps have to be repeated for them. If text which is referred-to is deleted then all cross-references to that text have to be found and any code or instructions nullified. Conversely, if new text is added which might be referred-to by existing text, then the entire document must be taken through steps a to e in order to add codes and instructions to existing text wherever it is required to make reference to the new text. The result is that the Hypertext file is static, in that items cannot be added or deleted without recompiling the entire Hypertext file.

A representation of a prior art Hypertext system is shown in FIG. 2. Once the authoring process is complete, the text portions are interrelated to one another in the manner described. To add new text, the author must decide which existing text portions should have links to the new text within the body of their text, and which existing text portions should have links from the body of new text. It can be seen that this is a time consuming, labour intensive operation.

SUMMARY OF THE INVENTION

It is, therefore, an object of the invention to provide a method of arranging and retrieving data in a computer which does not require the authoring process previously described. It is a more particular object of the invention to provide a method of cross-referencing data in a computer which does not require an author to identify and code links within the data. It is also an object of the invention to provide an effective method of searching data items within a system operating the invention.

These and other objects are accomplished by providing a method of associating portions of data stored in an information retrieval system, the system including:

- a display; and
 - a store for storing data as a plurality of data portions comprising data elements, each data portion having a unique reference name,
- the method comprising:
- (a) reading the data elements of a first data portion of said plurality of data portions;
 - (b) comparing said data elements of said first data portion with each of said reference names;
 - (c) determining where occurrences of said references correspond to data elements in said first data portion and determining a set of data portions for which such occurrences are found;
 - (d) associating said first data portion with each of said data portions in said set of data portions; and
 - (e) displaying said first data portion on said display.

The present invention may be implemented in a database in which each record comprises a data portion in the form of several pages of text relating to a particular topic, and a further field in the same record contains the unique reference

MS 067399

5,815,830

3

in the form of a topic name. Other data is also contained in other fields of the same record. A data portion, such as text on a particular topic, is displayed for the user to read and digest. Just prior to displaying the topic text, the invention compares the text to references, such as topic names for text, pictures, video and sound, for other data portions containing information on other topics. Upon display of the text to the user, the invention indicates that other topic names have been found within the topic text being displayed and read by the user, and that an association has been created between the text being read and the other topics. The user may then request any one of the topics found in the comparison be displayed. If the requested topic is a text topic, the process is repeated for the new topic. If the requested topic is a picture, or video topic, it will be shown or played without moving from the current text topic. The invention thus provides a structure for cross-referencing text which does not require manual authoring of cross-reference links. This reduces the time and effort required to produce, and maintain hypertext documents, and eliminates the need for the author to identify a word or phrase from which to cross-reference. There is also, therefore, no need for the author to embed codes to indicate the existence of a cross-reference, or to specify instructions for navigation to the referred to text.

The comparison operation may be a string search; a preferred comparison method is described later. The association process provides links between data elements, such as words or phrases and data portions, such as passages of text, and other data portions. One may consider that, as a data element within a first data portion is associated with a further data portion, then that first data portion is effectively associated to the further data portion. The invention and drawings will, therefore, be described in terms of associating, by linking, portions of text to other text portions or pictures or sound.

Generally, the operating environment of the present invention includes a general purpose computer system having a processor, a memory, a display and associated peripheral equipment such as disk drive storage or other storage medium. In particular, the invention is embodied in an application operating in a Windows™ environment on an IBM™ compatible personal computer. The invention is, advantageously, operable in a computer network such as workgroups, local-area networks (LAN) and wide-area networks (WAN), remotely and internationally. When operating in a network, the store of data in the form of a database may be centrally located, with each network user having access to the information therein. As the text stored is only compared, and associated, with topic names at the moment of display, the links between text portions are always up to date. If a topic is added to the database, viewing users have immediate access to that text. Any existing text which happens to refer to the new topic will automatically be associated and show the new topic as a cross-reference link. Conversely, if the author deletes a topic, no links to it will appear on the system.

The invention is effective with respect to textual data stored as separate topics; each topic comprising a few pages of text. However, that the invention applies to other types of data, such as: images, sound, video, executable files or other data.

BRIEF DESCRIPTION OF DRAWINGS

FIG. 1 is a flow diagram showing the prior art authoring process required to set up a hypertext system;

FIG. 2 shows the basic structure of links between text portions in prior art Hypertext systems;

4

FIG. 3 shows the basic structure of links between text portions in a system according to the invention;

FIG. 4 is a flow diagram showing the basic processes of the invention; and

FIG. 5 shows in detail a flow diagram of the preferred embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

The basic structure of data organisation in a system using the invention is shown in FIG. 3. The data portions 700 and reference names 600 are stored in the form of a database; with a reference name stored as a first field in a record, and the data portion to which that reference refers stored as a second field in that record. A reference name 600 is a unique, meaningful name which indicates the subject matter of the data portion to which it refers. The name may be a word, a phrase or other string indicative of the topic of the data portion. A data portion comprises pages of text on a particular topic, as well as any images, sound, video or executables. We refer to a record in the database, comprising topic data such as text pages, pictures or sound, reference name and any further fields of data, as a HyperNODE™. The database itself containing the records is referred to as a HyperDB™, and the application of the invention is known as XGL Hypertext VOYAGER™. The process of creating associations between data is named Auto Hyperlinking™.

Information is input into a system using the invention in the following manner. Text for a topic is either typed into the database, input as an ASCII text file, or input from other applications such as wordprocessors and database applications. The text may then be manipulated by the user by "cutting and pasting" the text in the usual manner for Windows™ applications. Alternatively, text may be cut and pasted from other Windows™ applications, such as wordprocessors. Various wordprocessors and database applications provide suitable text formats, such as: WordPerfect™, Word™ and DBase™. The author may also import pictures or sound from other Windows applications. Using any of these techniques the author may easily input text and pictures to construct the database of topics, with each topic stored as a separate record. The user also defines the reference name for each topic at this stage. An embodiment uses the same standards as the Paradox™ database application produced by Borland™, with which a database produced for the present invention is compatible. The reader is referred to the package Paradox™ for further details on the codes and the manner in which data is stored for Paradox™ compatibility. Other platforms such as IBM VSAM™, DBase™ and Oracle™ are equally suitable, and the invention is not limited to any particular format.

Once a database has been constructed, as described above, the user may interrogate the database using the invention, the comparison process 200, and association process 400 are shown in relation to the database in FIG. 3. Referring now to FIG. 4, the first step 100 is to read a first topic selected by the user. This selection is made by positioning a cursor on the display, using a mouse, in the usual manner for Windows™. The selection of the topic may be from a list of available topics. On selection, the first page of topic text is compared 200 to the other topic names in the database and then displayed. The comparison is conducted by automatically searching for the occurrence of topic names in the body of text of the first page of the first topic. A preferred searching technique to conduct the comparison is described later. On finding a match 300, the matched topic name

MS 067400

5,815,830

5

occurring within the text of the first topic is associated with the topic data of the topic to which the matched topic name refers.

Association 400 could involve simply indicating the existence of the related topic found in the search. However, the invention advantageously provides links, known as Auto_Hyperlinks™, meaning that the word or phrase in the text found to be a match with a topic name is highlighted on the display, and linked to the topic to which the topic name refers. The user may then jump to the associated topic by selecting the highlighted word or phrase in the first topic text, as in prior art Hypertext systems, or if the associated topic is a picture it is displayed on selection. The link is made with reference to the database which stores the topic text, reference name and other identifiers. Such identifiers note the location of the data for each topic, and provide the navigational links for the hypertext jumps.

Prior to displaying the first page of a topic, the comparison with the topic names is conducted for that page. The comparison for subsequent pages of that topic is also undertaken, preemptively while the first page of text is displayed, until the entire text for that topic has been compared, or the user has moved to a new topic 500. Text is thus linked a page at a time for the topic the viewing user has requested. The invention thus assumes that the user is likely to display the next page of the topic and so preemptively links the next pages while the user is reading the first page. The sequence is then repeated for the newly displayed topic as shown by steps 100 and 500 in FIG. 4.

The invention is set out in the flow chart of the invention of FIG. 5. The first page of text is read 100, the first word 201 selected and the next word of text 202 is scanned. If the word overflows the user predefined topic name length 203, then the database of topic names is searched in comparison to the concatenation buffer contents 208. If the word does not overflow 203, then the word is added to the buffer 204, and the word count is incremented by one 205. If the word ends in punctuation 206, the contents of the buffer is searched against the topic names 208. If no punctuation is present, and the predefined buffer word limit is not reached 207, the next word 202 is scanned. In this way, steps 202 to 207 successively scan words and adds them to the concatenation buffer until the predefined word limit is reached. At this point the concatenated phrase is searched against the database. If a topic is found 209, a link is created 211 (providing that the name does not refer to the present topic 210) and the process moves m words along, where m is the word count in the buffer. At this point the buffer is cleared in preparation for the next set of steps 202 to 207.

If no match is found with the buffer search term having the user defined maximum number of words, and a near match is found 213, the steps 213 to 218 check for plurals and, if still no match is found, successively removes the last word added to the buffer and compares the buffer contents with the list of topic names until a match is found. If no match is found, the system moves to the next word along 219 and starts the comparison again at 202. Once sufficient text has been processed to fit the size of the display window of the display, the processed text is displayed. Upon displaying the first page, the invention continues to repeat the above process for subsequent pages of the topic text.

A database has been created to cover the subject area: planets of our solar system. The database contains the following text and picture topics:

6

Earth	Io
Jupiter	Mars
Mercury	Neptune
Our moon	Planet
Pluto	Rings of Saturn.
Ring system	Satellites of Jupiter
Saturn	Solar System
The red spot (picture topic)	
Uranus	Venus

Topic being viewed by user: Planet

Text for topic planet:

There are ten major known planets in our solar system. Jupiter is the largest, with a diameter ten times that of the Earth. There are 12 satellites of Jupiter or moons, the most famous being Io. Jupiter which also has a faint ring system nowhere near as prominent as the rings of Saturn is famous for the red spot which is 3 times the diameter of the earth. Saturn is the second largest planet

The text for the topic planet is shown above. The words highlighted in bold and underlined have been automatically shown as hyperlinks. These words or phrases have been found to exist on the database as topics in their own right and the logical deduction from this is that there is more information available for them so they are automatically cross-referenced or Hyperlinked. If the user clicks the mouse on any of the hyperlinked words which refer to a text topic, they would automatically be taken to that topic and its associated topic text would be displayed with again any hyperlinks automatically found and highlighted as above. If the hyperlinked phrase "the red spot" is selected, the picture associated with the phrase is displayed without moving off the present text topic. In the above example the word planet is not hyperlinked because it relates to itself.

The following illustrates how concatenation and the finding of hyperlinks in the above text is achieved.

Concatenated Key	Operation performed
There are ten	First group of 3 words are concatenated and then a lookup is done with the key "There are ten" to see if a topic of this name exists. As it does not, the search continues with the next words.
are ten major	Next group of 3 words are concatenated
ten major known	And so on . . .
major known planets	
known planets in	
planets in our	
in our solar	
our solar system	
solar system.	Full stop causes concatenation of only these two words. As a match has been found, these words are marked as an Auto hyperlink. The process now continues, with the word immediately following the word "system".
Jupiter is the	Near match found, so continue with these words
Jupiter is	
Jupiter	Match found, so search continues after "Jupiter".
is the largest	
the largest with	
etc . . .	and so on until
10 satellites of	
satellites of Jupiter	Match found on this phrase
etc . . . until	
being are Io.	
are Io.	Full stop causes concatenation of two words

MS 067401

5,815,830

7

-continued

Concatenated Key	Operation performed
Io.	only
Jupiter also has	Match found
	Near match found so continue to try these words
Jupiter also	
Jupiter	Match found
as the rings	
the rings of	
rings of Saturn.	Match found
the red spot	Match found, and end of text.

In this example the user defined maximum number of words to be concatenated is three. The searching process described above is an effective way of comparing text for matching strings comprising several words.

It should be noted that, whilst the invention has been described with reference to textual data, it is clear that the invention is applicable to other types of data. In essence, the invention provides a dynamic structure for relating information in a store, and displaying the information, along with links between the data found by the invention. The data portions can be any suitable data, particularly alphanumeric data. For non-alphanumeric data, such as images, video or executables, the data has a reference name as with textual data portions. This reference name may be searched within text of other data portions and linked in the manner described. A phrase in a document may, therefore, refer to an image name which, when selected, automatically displays the image having that name. If the referred to data portion is an executable file, that executable will be executed on selection of its name from the highlighted text. Many applications other than simple cross-referencing may thus be envisaged. Executable management in this way provides a useful system for a user to perform operations such as copying or deleting data, or other disk and memory management functions.

In the embodiment described the invention is particularly effective as the association of data portions is undertaken just prior to display of the selected topic. This ensures that the system is always up to date unlike prior manually authored systems which are only updated when the author manually adds the links. This aspect allows multiple users to add topics to the database without the need to be aware of the topics already stored on the system. In the context of a news system, for example, individuals may add news from different parts of the world onto the system without the need of communicating with one another. Since the association of topics occurs just prior to display, the database thus formed is always instantly up to date requiring no manual authoring or compilation.

Further rules of topic association other than simply searching for identical and similar word strings or using the concatenation search previously described may also be used. For example linguistic rules may be incorporated so that the phrase "moons of jupiter" within a portion of text is associated with a topic having the topic name "jupiter's moons". Other linguistic rules may also be applied such as associating a topic with the topic name "people" with an occurrence of the word "person". These and other linguistic rules are within the scope and spirit of the invention.

It should also be noted that the topic names and text portions do not have to be stored in a database, and could be stored as separate files, or otherwise. A list of topic names could also be stored as a single file, with the topic text stored as a single flat file. The invention may thus be implemented

8

in a variety of environments and platforms other than the embodiment described. As well as implementing the invention to run on a computer to provide hypertext information to a user on a computer, the invention can be used to provide hypertext information on a television. Some examples of useful television applications are: a news service providing up to date hypertext pages of news, a TV programme information service or an educational/exploratory service.

Taking the news service as an example: with news coming in all the time, a set of news pages could be dynamically maintained by several authors using the invention, all updating the same HyperDB. On detecting that a topic text has been changed the invention would hyperlink the text pages of that topic in the same way as described earlier. But instead of displaying the pages, the invention would pass the newly hyperlinked topic text to a Text Transmission Control Unit (TTCU) or some such similar processor. The TTCU would store the said pages, which would include appropriate indicators to distinguish hypertext words and phrases and also a news page number associated with each such hypertext reference. The TTCU, when all pages for a topic have been stored, would broadcast the said topic text as a number of pages. In normal operation the TTCU would be broadcasting all news pages continually so that any TV user could be receiving any page at any time. If a new topic has been added or one deleted from the database, the invention on detecting such events (as is currently done when the invention is run on a network) would automatically hyperlink, as described before, all the topics on the news database and pass each page to the TTCU which would again store them. Specifically in this event (adding/deleting of entire topics) the TTCU would receive and store all pages of the news database before it started to broadcast any of the new pages. This is because the hypertext linking would now relate to the entire set of pages comprising the news database including added or deleted topics.

If a TV user were to select a particular hypertext link the news page associated with the selected link would simply be displayed on the TV. This "jump" technology already exists in that you can display different pages of teletext or similar at will. Such similar technology could be used to achieve the jump associated with the hypertext link within the news pages, but with the additional advantages provided by the association process of the invention. Alternatively, a small piece of software (an extension of the implementation of the invention) would reside in all TV's capable of receiving hypertext pages from the invention.

A third alternative is that the same or similar piece of jump software from the invention could be implemented on the newly emerging TV-PC technology. Such jump software could be made freely available as part of the normal software provided with the TV-PC technology. Selection of a conventional PC or similar is typically done using a mouse by pointing and clicking. Televisions do not have an equivalent device, though a mouse roller ball and a selection button could be incorporated into the remote control. The usage of these would be conventional; the user would manipulate the roller ball to position a cursor on the screen and then press the button to select the item where the cursor rests.

Taking the television programme service as a second example: A set of television programme pages could be dynamically maintained by a number of authors. The pages would be equivalent to published journals like the BBC Radio/TV Times. This example would work in the same way as the example on the news service with the addition of a further feature.

The additional feature requires the use of a TV-PC instead of just a TV. The invention will further enable the authors to

MS 067402

5,815,830

9

include in the programme information a Unique Alphanumeric Code (UAC) associated with each television, radio and future multi-media/virtual-reality/cyber-space programmes. The invention would enforce uniqueness when authors assigned a UAC to a programme. The invention when hyperlinking would for every found hyperlink include, as well as the page number as before, the UAC if a UAC existed for the reference hyperlink. Such pages as before would not be displayed but passed to the TTCU. The TTCU would broadcast hypertext pages as normal but also include the UAC (where one exists) with every hypertext reference.

In this application, the TV-PC user will be able to not only jump to different pages of hypertext information, but also to indicate to the TV-PC to store any hypertext reference and its UAC in a Personal Programme Schedule (PPS). The PPS could then be reviewed by the TV-PC user and each programme entry could be marked by the user to indicate actions that the TV-PC is to take such as "get me to watch/hear", "record on video or audio or CD or PC memory", or "censor" (stopping children watching inappropriate programmes). Once in the system the software would allow the user to add/remove further items and would be able to warn of scheduling conflicts, amount of tape/memory required to record selected programmes, etc. This provides the user with far greater control over the watching and recording of programmes. It assumes the TV-PC technology will also be connected to a Hi-Fi system which includes a radio and tape recorder.

It should be understood that the foregoing description of the present invention is illustrative only. Thus, although only a few examples of the invention have been described in detail, it is clear that the features of the present invention may be adapted without departing from the spirit of the invention.

What is claimed is:

1. A method of searching similar strings in a system for associating portions of text stored in an information retrieval system, the system including:

a display; and

a store for storing data as a plurality of portions of text, each portion of text relating to a particular topic, and having a unique topic name stored therewith;

said method of searching comprising:

comparing a first concatenated string of words in said text with said topic names;

determining whether a topic name matches said concatenated string;

removing one of the words from the concatenated string to form a new string if no match is found; and

comparing the new string with said topic names.

2. A method of associating portions of data stored in an information retrieval system, the system including:

a display; and

a store for storing data as a plurality of data portions comprising data elements, each data portion having a unique reference name,

the method comprising:

(a) reading the data elements of a first data portion of said plurality of data portions;

(b) comparing said data elements of said first data portion with each of said reference names;

(c) determining where occurrences of said references correspond to data elements in said first data portion and determining a set of data portions for which such occurrences are found;

10

(d) associating said first data portion with each of said data portions in said set of data portions; and

(e) displaying said first data portion on said display.

3. A method of associating portions of data stored in an information retrieval system according to claim 2, wherein the data portions and references are stored as records in a database.

4. A method of associating portions of data stored in an information retrieval system according to claim 3, wherein each said record comprises a first field for reference names, and a second field for data portions.

5. A method of associating portions of data stored in an information retrieval system according to claim 2, wherein at least one of the data portions comprises an image.

6. A method of associating portions of data stored in an information retrieval system according to claim 2, wherein at least one of the data portions comprises video data.

7. A method of associating portions of data stored in an information retrieval system according to claim 2, wherein at least one of the data portions comprises an executable.

8. A method of associating portions of data stored in an information retrieval system according to claim 2, wherein at least one of the data portions comprises sound data.

9. A method of associating portions of data stored in an information retrieval system according to claim 2, wherein the comparison to find corresponding references for said data elements comprises searching for a reference which is identical to one of said data elements.

10. A method of associating portions of data stored in an information retrieval system according to claim 2, wherein the system comprises a buffer, said method further comprising selecting a predetermined number of adjacent data elements in said first data portion, storing said adjacent data elements in said buffer, and comparing the contents of said buffer to said references.

11. A method of associating portions of data stored in an information retrieval system according to claim 10, wherein one of said adjacent data elements stored in said buffer is removed from said buffer if the contents of said buffer is not identical to one of said references, and said buffer contents is subsequently compared to said references.

12. A method of associating portions of data stored in an information retrieval system according to claim 11, wherein said buffer has a user defined buffer limit which determines the predetermined number of data elements stored.

13. A method of associating portions of data stored in an information retrieval system according to claim 2, wherein: at least one of said data portions comprises alphanumeric data on a particular topic;

said data elements comprise alphanumeric strings; and

said reference of said data portions are alphanumeric topic names.

14. A method of associating portions of data stored in an information retrieval system according to claim 2, wherein: at least one of said data portions comprises pages of text on a particular topic;

the data elements comprise text strings; and

the reference of said at least one of said data portions is a textual topic name.

15. A method of associating portions of data stored in an information retrieval system according to claim 14, wherein there are plurals within said pages of text and the comparison searches for singulars of said plurals in said references.

16. A method of associating portions of data stored in an information retrieval system according to claim 14, wherein the comparing operation comprises scanning said pages of text for words and phrases and punctuation is temporarily stripped out.

MS 067403

5,815,830

11

17. A method of associating portions of text stored in an information retrieval system, the system including:

a display; and

a store for storing data as a plurality of data portions at least one of which is text, each portion relating to a particular topic, and having a unique topic name stored therewith,

the method comprising:

(a) reading said at least one portion of text of said plurality of portions;

(b) comparing the text of said at least one portion of text with each of said topic names;

(c) determining a set of portions of data for which occurrences of said topic names are found in said at least one portion of text;

(d) associating said at least one portion of text with each of said portions in said set of portions of data; and

(e) displaying said at least one portion of text.

18. A method of associating portions of text stored in an information retrieval system according to claim 17, wherein said association includes indicating that occurrences are found by highlighting said occurrences in said first portion of text.

19. A method of associating portions of text stored in an information retrieval system according to claim 18, wherein said association provides a link to said portions of data in said set of portions of data such that selection of one of said highlighted occurrences displays, plays or executes the data portion having the reference of said occurrence.

12

20. A method of associating portions of text stored in an information retrieval system according to claim 17, wherein said topic names have related topic identifiers.

21. A method of associating portions of text stored in an information retrieval system according to claim 18, wherein said topic names have related topic identifiers and said link is created with reference to said identifiers.

22. A method of associating portions of text stored in an information retrieval system according to claim 17, wherein said data portions and topic names are stored as records in a database.

23. A method of associating portions of text stored in an information retrieval system according to claim 17, wherein said database comprises a first field and a second field, and said first field comprises said topic names and said second field comprises said data portions.

24. A method of associating portions of text stored in an information retrieval system according to claim 17, wherein the system comprises a buffer, said method further comprising selecting a predetermined number of adjacent words in said at least one portion of text, storing said adjacent words in said buffer, and comparing the contents of said buffer to said topic names.

25. A method of associating portions of text stored in an information retrieval system according to claim 24, wherein one of said adjacent words stored in said buffer is removed from said buffer if the contents of said buffer is not identical to one of said topic names, and said buffer contents is subsequently compared to said topic names.

* * * * *